

高等学校计算机语言应用教程

Delphi 7 应用教程

童爱红 张 琦 胡光兵 编著

清华大学出版社

北京交通大学出版社

• 北京 •

内 容 简 介

本书对 Delphi 7 语法和程序设计思想进行了全面的阐述,共分 14 章,全面讲解了面向对象的程序设计概念、Delphi 7 的数据类型与表达式、Delphi 7 程序设计语句、Delphi 的组件与窗体、数组程序设计、过程与函数程序设计、文件程序设计、组件和 DLL 开发技术、数据库程序设计和多媒体程序设计等内容。

本书从教学实践的角度出发,立足于提高学生的程序设计应用能力,全书理论分析透彻严谨,实例丰富生动,内容由浅入深,能快速地引导学生进入 Delphi 7 编程世界。本书可作为高等院校程序设计课程教材,也可作为广大希望掌握 Delphi 7 编程的程序设计人员的参考用书。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

Delphi 7 应用教程 / 童爱红, 张琦, 胡光兵编著. —北京: 清华大学出版社; 北京交通大学出版社, 2004.11

(高等学校计算机语言应用教程)

ISBN 7-81082-390-6

I. D… II. ①童… ②张… ③胡… III. 软件工具-程序设计-高等学校-教材
IV. TP311.56

中国版本图书馆 CIP 数据核字(2004)第 085187 号

责任编辑: 谭文芳

出版者: 清华大学出版社 邮编: 100084 电话: 010-62776969

北京交通大学出版社 邮编: 100044 电话: 010-51686045, 62237564

印刷者:

发行者: 新华书店总店北京发行所

开 本: 185×260 印张: 21 字数: 534 千字 附光盘 1 张

版 次: 2004 年 11 月第 1 版 2004 年 11 月第 1 次印刷

书 号: ISBN 7-81082-390-6/TP·143

印 数: 1~5000 册 定价: 34.00 元(含光盘)

目 录

第 1 章 Delphi 7 程序设计入门	1
1.1 理论知识	1
1.1.1 算法与程序设计	1
1.1.2 面向对象程序设计基础	2
1.1.3 Delphi 7 概述	4
1.1.4 Delphi 7 的集成开发环境	4
1.1.5 利用 Delphi 开发应用程序的方法步骤	7
1.1.6 Delphi 7 程序的组成	11
1.2 典型实例	13
1.3 上机练习	14
课后考场	15
第 2 章 基本数据类型与表达式	17
2.1 理论知识	17
2.1.1 基本数据类型	17
2.1.2 Delphi 7 中的标识符与保留字	20
2.1.3 Delphi 7 中的常量与变量	21
2.1.4 Delphi 7 中的运算符与表达式	23
2.1.5 Delphi 7 的语句书写规则与赋值语句	25
2.1.6 Delphi 7 的常用函数与过程	26
2.2 典型实例	31
2.3 上机练习	33
课后考场	34
第 3 章 基本程序设计语句	36
3.1 理论知识	36
3.1.1 基本的顺序结构语句及其应用	36
3.1.2 选择结构语句及其基本应用	37
3.1.3 循环结构语句及其基本应用	43
3.2 典型实例	46
3.2.1 典型实例一	46
3.2.2 典型实例二	48
3.2.3 典型实例三	49
3.2.4 典型实例四	51
3.2.5 典型实例五	52

3.3 上机练习	54
3.3.1 上机练习一	54
3.3.2 上机练习二	54
课后考场	55
第4章 数组程序设计	59
4.1 理论知识	59
4.1.1 数组的概念	59
4.1.2 一维静态数组的定义与使用	59
4.1.3 二维数组及多维数组的定义与使用	62
4.1.4 动态数组的定义与使用	64
4.2 典型实例	67
4.2.1 典型实例一	67
4.2.2 典型实例二	69
4.3 上机练习	70
4.3.1 上机练习一	70
4.3.2 上机练习二	72
课后考场	74
第5章 过程与函数	76
5.1 理论知识	76
5.1.1 过程与函数的概念	77
5.1.2 过程的定义与调用	77
5.1.3 函数的定义与调用	79
5.1.4 参数的传递	81
5.1.5 子程序的嵌套与递归	83
5.2 典型实例	87
5.2.1 典型实例一	87
5.2.2 典型实例二	89
5.3 上机练习	90
5.3.1 上机练习一	90
5.3.2 上机练习二	91
课后考场	93
第6章 用户自定义类型	95
6.1 理论知识	95
6.1.1 枚举类型的定义与使用	95
6.1.2 子界类型的定义与使用	97
6.1.3 集合类型的定义与使用	98
6.1.4 记录类型的定义与使用	101
6.2 典型实例	104
6.2.1 典型实例一	104

6.2.2 典型实例二·····	106
6.3 上机练习·····	107
6.3.1 上机练习一·····	107
6.3.2 上机练习二·····	109
课后考场·····	111
第7章 常用组件的使用 ·····	113
7.1 理论知识·····	113
7.1.1 文本类组件的使用·····	113
7.1.2 按钮类组件的使用·····	121
7.1.3 列表类组件的使用·····	124
7.1.4 TTimer 时钟组件的使用·····	128
7.1.5 对话框组件的使用·····	129
7.1.6 TImage 组件·····	132
7.1.7 菜单组件·····	133
7.1.8 TTabControl 组件和 TPageControl 组件的使用·····	137
7.1.9 TScrollBar、TTrackBar 和 TProgressBar 组件的使用·····	141
7.1.10 TPanel 组件和 TGroupBox 组件·····	143
7.1.11 工具栏组件与状态栏组件·····	144
7.2 典型实例·····	144
7.2.1 典型实例一·····	144
7.2.2 典型实例二·····	147
7.3 上机练习·····	149
7.3.1 上机练习一·····	149
7.3.2 上机练习二·····	150
课后考场·····	151
第8章 Delphi 7 的文件系统 ·····	154
8.1 理论知识·····	154
8.1.1 文件的基本概念·····	154
8.1.2 Delphi 7 中的文件类型及文件类型变量的定义·····	156
8.1.3 文本文件的使用·····	157
8.1.4 记录文件的使用·····	163
8.2 典型实例·····	170
8.2.1 典型实例一·····	170
8.2.2 典型实例二·····	172
8.3 上机练习·····	173
8.3.1 上机练习一·····	173
8.3.2 上机练习二·····	175
课后考场·····	177
第9章 应用程序界面设计技术 ·····	179

9.1 理论知识	179
9.1.1 多窗体程序的设计	179
9.1.2 SDI 应用程序设计技术	182
9.1.3 MDI 应用程序设计技术	183
9.1.4 变量的作用域	186
9.2 典型实例	188
9.3 上机练习	191
课后考场	195
第 10 章 DLL 应用编程	197
10.1 理论知识	197
10.1.1 DLL 概述	197
10.1.2 DLL 编写	199
10.1.3 加载 DLL 的方法	202
10.1.4 调用 DLL 中的过程和函数	204
10.1.5 在 DLL 中实现窗体重用	208
10.2 典型实例	212
10.3 上机练习	215
课后考场	218
第 11 章 组件开发技术	219
11.1 理论知识	219
11.1.1 组件与组件技术概述	219
11.1.2 确定组件基类	221
11.1.3 创建组件单元	223
11.1.4 创建包工程	224
11.1.5 在组件中添加属性	225
11.1.6 在组件中添加事件	231
11.1.7 调试组件	234
11.1.8 制作组件图标和发布组件	235
11.2 典型实例	235
11.3 上机练习	238
课后考场	240
第 12 章 图形图像编程	242
12.1 理论知识	242
12.1.1 TCanvas 对象的使用	242
12.1.2 TGraphic 对象的使用	250
12.1.3 TPicture 对象的使用	250
12.1.4 TBitmap 对象的使用	252
12.1.5 Delphi 中的图形图像组件	252
12.2 典型实例	255

12.2.1 典型实例一	255
12.2.2 典型实例二	257
12.3 上机练习	259
12.3.1 上机练习一	259
12.3.2 上机练习二	261
课后考场	263
第 13 章 多媒体应用程序开发	265
13.1 理论知识	265
13.1.1 多媒体的概念	265
13.1.2 TAnimate 组件的使用	266
13.1.3 TMediaPlayer 组件的使用	269
13.2 典型实例	274
13.2.1 典型实例一	274
13.2.2 典型实例二	277
13.3 上机练习	279
13.3.1 上机练习一	279
13.3.2 上机练习二	280
课后考场	282
第 14 章 数据库应用开发	283
14.1 理论知识	283
14.1.1 数据库的基本概念	283
14.1.2 利用数据库桌面创建数据库	284
14.1.3 利用 BDE 组件开发数据库应用程序的模式	289
14.1.4 TTable 组件	290
14.1.5 TDataSource 组件	294
14.1.6 Data Controls 组件	294
14.1.7 SQL 语言	298
14.1.8 TQuery 组件	300
14.2 典型实例	303
14.2.1 典型实例一	303
14.2.2 典型实例二	308
14.3 上机练习	311
14.3.1 上机练习一	311
14.3.2 上机练习二	313
课后考场	318
附录 A Delphi 中的虚拟键代码及对应的键	321
参考文献	323

前 言

关于 Delphi 程序设计的相关教材在市场上有一些,通过研究发现这些教材基本上都有这样一个通病:即为写 Delphi 而写教材。由于 Delphi 软件功能强大、内容丰富、组件众多,给开发应用程序带来了很大便利,任何一方面的内容都足以写成厚厚的一本书,把握不好就有可能出现把握不住要点、缺乏系统性、只讲 Delphi 而没有涉及程序设计精髓,或者干脆就成了纯粹的编程技巧说明,不太适合作为程序设计课程的入门教材。正是在此背景下,我们编写了这本《Delphi 7 应用教程》,目的是为了引导学生快速地掌握程序设计并高效地进入 Delphi 编程世界。

本书的编写人员都有多年从事程序设计教学的一线教学经验,对程序设计的教学把握较为独到,能够预料到学生在学习过程中可能遇到的困难并加以解决。同时本书的编写人员均有教材编写的经验,具有很强的敬业精神,编写的教材有助于提高学生的学习效率。

本书的总体编写思路如下。

1. 全书共 14 章,全面讲解了 Delphi 程序设计语言的各个部分,特别加强了对数组、过程与函数、界面设计、文件、多媒体、数据库等程序设计的重点、难点和具有较高实用价值的内容的引导和剖析。

2. 每章均分 4 个部分进行编写:“理论知识”部分简明扼要地讲解本章的主要理论,并通过小的实例进行深化理解;“典型实例”部分通过分析一些精心挑选和编制的典型实例,强化学生的编程能力;“上机练习”部分挑选一些具有实用价值的上机练习题,加以剖析并给出部分程序代码,引导学生在上机练习中提高应用能力;“课后考场”部分设计了一套试题,方便学生进行自我测试。

3. 实例引导。本书的每一章都有丰富的实例,有的实例具有较强的趣味性,易引起学生的兴趣,激发学生对程序设计的喜好。

与现有的教材相比,本书具有以下特色。

1. 重点突出。本书没有罗列大量的语言成分,不介绍较琐碎或不太常用的属性、指令和方法,而是针对 Delphi 软件的特点,较详细地介绍 Delphi 的主要语言成分,重点讲述 Delphi 程序设计的概念和方法。

2. 不为写 Delphi 而写教材。始终贯彻为写程序设计教材而写教材,Delphi 只是选择的一门工具语言。因此本书将重点放在程序设计的基础上,放在程序设计教材的共性上,如结构化和面向对象的程序设计方法、一些简单常用的算法、三种结构的程序设计、数组程序设计、过程与函数程序设计、文件编程、多媒体编程、DLL 编程等要素上,而不是仅着重于 Delphi 的强大功能和组件的用法技巧上。本书力争达到这样的目标:通过本书的学习使学生能够掌握程序设计的概貌,进入程序设计的大门,而不是仅仅掌握 Delphi 的语法与组件使用。

3. 在编写风格上注重学生动手编程能力的培养。针对学生普遍认为程序设计难学的问题,本书将不再讲解高深难懂的理论,而是强调通过实例学编程。通过精选有趣的实例,讲解实

例的实现过程，激发学生的编程兴趣，引导学生一步一步地步入程序设计的大门。

本书由童爱红、张琦、胡光兵具体编写，最后由博士生导师张琦教授对全书进行了审阅并定稿。

本书的配套光盘包含全书的所有例题、习题源代码和可执行文件，所有的程序都在 Windows XP 平台和 Delphi 7 环境下调试通过并经过严格测试。另外，本书配套的电子教案可以在北京交通大学出版社网站 <http://press.bjtu.edu.cn> 上下载。

在本书的编写过程中，得到了解放军理工大学计算机与指挥自动化学院黄松副教授的指导和帮助，得到了解放军理工大学工程兵工程学院计算机应用教研室全体老师的指导与帮助，在此表示衷心的感谢。同时对参加本书资料收集、程序测试和文稿校对的侯太平、汪刚等同志表示衷心的感谢。编者参阅了大量文献资料及网站资料，在此也一并表示感谢。

虽然我们力求完美，力创精品，但由于水平有限，书中难免有疏漏和错误等不尽人意之处，还请广大读者不吝赐教。

编 者
2004 年 9 月

第 1 章 Delphi 7 程序设计入门

本章要点

- ▮ Delphi 7 的集成开发环境
 - ▮ 面向对象程序设计的概念
 - ▮ 设置对象属性、调用对象方法和编写程序代码的方法
 - ▮ Delphi 单元文件和项目文件的结构
 - ▮ 利用 Delphi 7 开发应用程序的一般步骤
-

1.1 理论知识

1.1.1 算法与程序设计

1. 算法

程序设计是与算法紧密联系在一起。广义地说，算法就是解决一个问题或完成一个任务所经历的确定的有限的步骤。例如，某个人要买一台价格 5000 元左右的品牌电脑，在购买之前，他可能已有意或无意地制订了如下的计划：①打电话或上网查询或向别人咨询了解市场上有哪些品牌电脑；②根据电脑的性价比和各品牌的售后服务等因素决定买哪一种品牌电脑；③决定去哪一家商场买电脑；④去银行取款；⑤去商场买电脑；⑥电脑买回家并向家人汇报。以上买电脑的过程就已体现了程序设计的实质——算法！从上例中可以看到算法具有以下 5 个特点。

（1）输入性。一般完成一个问题都要给定一定的初始信息或数据，有了初始信息或数据算法才能一步一步地往下执行。如上例中的银行取钱，就可以看做对算法的输入。

（2）输出性。算法执行完成后，应该输出需要的信息。如上例中，电脑买回家后要汇报情况就可看做算法的输出。

（3）可执行性。算法的每一步都必须是可以执行的。

（4）有穷性。一个算法必须能用有限的步骤完成。

（5）确定性。算法中的每一步都必须是确定的，不能是模棱两可的。

程序设计中所讲的算法就是指为了使计算机完成某个任务而经过的确定的有限的步骤。

2. 程序设计

下面介绍一个程序设计的例子。

【例 1-1】 两个数 a 和 b，如果 a 大于 b，就交换它们的值，否则不交换。具体的算法步骤如下：

- (1) 比较 a 和 b 的大小;
- (2) 如果 a 小于等于 b, 转到第 (5) 步;
- (3) 如果 a 大于 b, 转到第 (4) 步;
- (4) 将 a 赋值给 t, 将 b 赋值给 a, 将 t 赋值给 b;
- (5) 程序结束。

上面的算法是用自然语言描述的, 计算机并不能直接执行用自然语言描述的算法, 要让计算机执行, 必须把由自然语言描述的算法的每一步转换成计算机语言的语句, 这个过程就是程序设计。上面的算法可用 Delphi 语言描述如下。

```
var
  a,b,t:Real;      //定义三个实型变量
begin
  //.....
  if (a>b) then      //如果 a 大于 b (否则, 直接转到最后一个“end;”语句, 程序结束)
  begin              //执行下面的复合语句
    t:=a;            //将 a 赋值给 t
    a:=b;            //将 b 赋值给 a
    b:=t;            //将 t 赋值给 b (从而完成 a 和 b 交换的功能)
  end;
end;                //程序结束
```

至此, 一个简单的程序已经编制成功了。

1.1.2 面向对象程序设计基础

高级语言刚出现的时候, 使用的是面向过程的程序设计, 这种方法发展到了一定的阶段就产生了“软件危机”。为消除软件危机, 出现了面向对象程序设计 (Object-Oriented Programming, OOP), 并诞生了面向对象的程序设计语言, 如 C++ 等。为了便于用户开发应用程序, 又出现了可视化程序设计语言, 如 Visual Basic, Delphi 等。面向对象与可视化程序设计的一般模型可用图 1-1 表示。

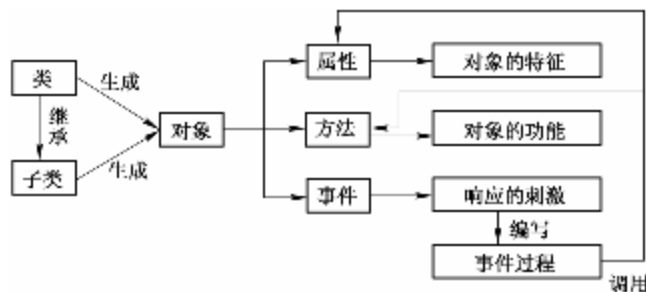


图 1-1 可视化程序设计方法示意图

1. 类与对象

类是对象的模板, 它定义了对对象的特征和行为规则, 对象是通过类产生的, 类和对象都

由惟一的名字进行标识, 分别是类名和对象名。可以把类想像成房屋结构图, 而对象就是按照这房屋结构图建成的房屋。可以根据一个类构造很多个对象, 由一个类产生的对象基本上都具有同样的特征和性能。在可视化程序设计语言中, 通常把一些常用的界面元素或功能实现预先定义成类, 使用时就可以直接拖动它们到界面上产生对象。在程序中也可以通过系统提供的类的要素进行修改, 衍生出许多子类, 从而可以创建出许多可视化的对象。

2. 属性

客观世界中的对象都具有一些特征, 并通过特征相互区分。如学生的特征有学号、姓名和专业, 并通过姓名或学号来区分。在面向对象程序设计中, 用属性来刻画对象的特征, 定义对象的外观。具体地说, 属性是类或对象的一种成分, 它反应类创建的对象特征, 如对象的名称、大小、标题, 等等。可视化语言中, 类或对象的属性是由类似的变量组成的, 每个属性都有自己的名字及一个相关的值, 标准组件的属性名基本上都是系统规定好的。在学习 Delphi 的过程中要注意记住属性名和理解属性名的含义。Delphi 中的每个系统提供的类都有一系列的属性, 在许多场合都可以通过可视化的手段或程序设计的方法改变属性的值。

3. 方法与事件

客观世界中的对象都具有一定的功能, 并都能对外界的特定刺激做出反应。反映在面向对象程序设计中这种对象功能就是方法, 能够响应的刺激就是事件。即方法是对象具有的功能, 而事件是对象能够响应的刺激。方法与事件是类的成分, 它们共同决定了对象的行为特征。实际上方法就是封装在类里面特定的过程或函数, 这些过程或函数的代码, 一般用户很难看到, 这就是类的“封装性”。方法由方法名来标识, 标准组件的方法名一般也是系统规定好了的。在 Delphi 中, 所说的组件其实就是一种类, 一般每个类都具有一系列的标准方法, 如 Form 类具有 Show, Close, Hide 等方法。

事件可视为对对象的一种操作。如在程序运行中, 对某个对象用鼠标单击一次, 这就产生了一个该对象的一个“单击(Click)事件”。事件由事件名标识, 组件的事件名也是系统规定好的。在学习 Delphi 的过程中, 也要注意记住事件名、含义及其发生场合。在 Delphi 中, 事件一般都是由用户通过输入手段或者是系统某些特定的行为产生的, 输入手段如鼠标在某对象上单击一次, 产生一个 Click 事件, 到达系统的特定行为定时器的时间间隔, 会发生定时器对象的 Timer 事件。

4. 事件驱动的程序设计

面向对象程序设计语言的基本程序设计模式是事件驱动。通过该方法设计的应用程序, 程序的执行是由事件驱动的, 一旦程序启动后就根据发生的事件执行相应的程序代码(事件过程); 如果无事件发生, 则程序就空闲着, 等待事件的发生, 此时用户也可以启动其他的应用程序。因此在这种程序设计模式下, 程序员只需考虑发生了某事件时, 系统该做什么, 从而设计出相应的事件过程代码, 事件过程代码通常很短, 也易编写。

5. 可视化编程的一般步骤

利用可视化语言设计程序的一般方法可归结成以下几个步骤。

(1) 利用系统给定的可视化类设计出程序运行界面(窗口)。一般的可视化程序设计软件都提供了空白窗口和可视化工具栏, 在设计程序时, 可以把可视化工具拖动到空白的窗口中, 就可以像“画图”一样轻松地完成程序运行界面的设计。

(2) 设计窗口和可视化工具的属性。属性控制了对对象的外观和表现形式, 通过属性设置

使对象的外观更符合程序员的要求。

(3) 编写事件过程代码。根据程序实现的功能要求编写相应事件的事件过程代码，在事件过程代码中可以设置对象的属性，从而改变对象的外观和表现形式，可以调用对象的方法来实现某种功能。

1.1.3 Delphi 7 概述

Delphi 是 Borland 公司推出的非常理想的可视化程序设计环境，特别适合于快速规范地开发 Windows 应用程序。从 Borland 公司推出 Delphi 1.0 至今，Delphi 已经发展了 7 代产品，每一代都是伴随 Windows 操作平台的升级而升级。用户可以使用 Delphi 轻松地进行各种复杂应用程序的开发。

1.1.4 Delphi 7 的集成开发环境

Delphi 程序设计、运行和测试等工作在 Delphi 的集成开发环境（Integrated Development Environment, IDE）里进行，Delphi 的 IDE 是一个功能非常强大、使用非常方便的开发环境，主要包括主窗口、窗体设计器、代码编辑器、代码浏览器和对象观察器 5 大部分。

1. Delphi 7 的主窗口

启动 Delphi 7，屏幕上会出现如图 1-2 所示的集成开发环境窗口。其中上半部分为菜单栏、工具栏和组件板。

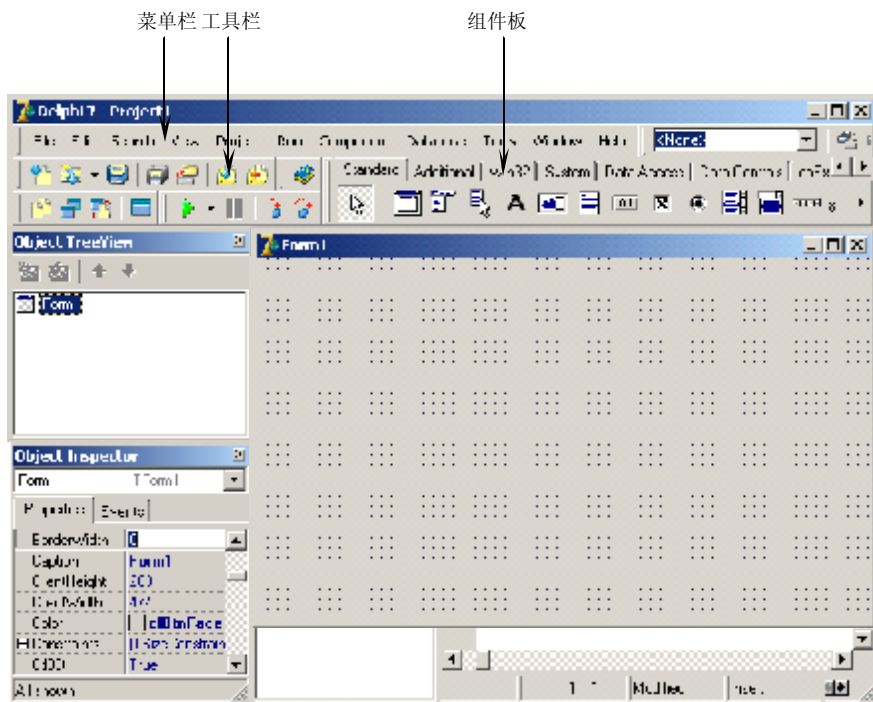


图 1-2 Delphi 7 的集成开发环境窗口

(1) 菜单栏

菜单栏包括【File】、【Edit】、【Search】、【View】、【Project】、【Run】、【Component】、【Database】、【Tools】、【Window】和【Help】共 11 个下拉菜单，包含 Delphi 所有的命令和功能。单击菜单栏中的任一菜单名，即可打开该菜单名对应的下拉菜单。表 1-1 简单介绍了菜单栏的各种菜单的基本功能。

表 1-1 菜单栏的各种菜单功能简介

菜 单 名	基 本 功 能
File	提供文件操作命令
Edit	提供与编辑操作有关的命令，如剪切、复制、粘贴和删除等
Search	提供查找、替换代码等操作命令
View	用来关闭或打开集成环境中的各种工具窗口，配置集成环境
Project	用于项目管理和设置
Run	提供应用程序的运行、调试等命令，例如设置断点等
Component	用于建立、安装和设置组件
Database	提供开发数据库的各种工具
Tools	用于开发环境的设置和提供辅助开发工具
Window	用来在打开的各个窗口之间转移焦点
Help	提供帮助信息

(2) 工具栏

工具栏的按钮一般是与菜单栏中的一些常用菜单命令对应起来的，并以直观的图标反映命令的功能，用于快速执行命令。另外，只要将鼠标指针停留在任一工具按钮，就会出现文字提示，用来提示该工具按钮的功能。

(3) 组件板

Delphi 7 组件板上的选项卡里一共有 32 个标签（页），它们包含了 Delphi 的所有组件（又称控件），其中每个标签（又叫选项卡）分别对应一类不同的组件。选择组件的方法是：首先用鼠标单击该组件所在的页（熟悉后就知道常用组件在哪个页下），然后单击需要的组件。把组件添加到窗体上的方法是：选择了一个组件后，再在窗体适当位置单击并拖动鼠标，松开鼠标时将会在窗体上创建一个组件。图 1-3 中选择的是【Standard】选项卡，但没有选择具体的组件。图 1-4 是选择【System】选项卡下的时钟组件的情况。



图 1-3 Delphi 主窗口中的组件板

注意：有如下 3 种方法向窗体添加组件。

- 1 单击组件板上的组件，然后在窗体的适当位置单击并拖动鼠标。
- 1 直接双击组件，可将该组件添加到窗体的中心位置。

- 1 按住 Shift 键不放，单击组件板上所需的组件，该组件将出现蓝色边框，同时对对象选择按钮将弹起，此时可在窗体上连续拖动鼠标多次以画出多个该组件。用鼠标单击对象选择按钮可取消该组件的连续添加操作。



图 1-4 选择 System 选项卡中的 Timer 组件

2. Delphi 7 的窗体设计器

Delphi 的窗体设计器用来可视化地设计应用程序窗口，是放置组件的容器。当 Delphi 启动完成后，就自动生成一个空白窗体，如图 1-5 所示。窗体由标题栏（通常包括控制菜单、最小化、最大化和关闭按钮）、工作区和可变大小的边界组成。

注意：改变窗体位置或窗体大小有如下 3 种方法。

- 1 鼠标左键按住窗体标题栏，可以把窗体拖动到任意位置。
- 1 鼠标指针指向边界，将会出现双向箭头，此时拖动鼠标可改变窗体大小。
- 1 在对象观察器中改变窗体的 Height 和 Width 属性值可以改变窗体的大小，改变 Top 和 Left 属性值可以改变窗体的左上角相对于屏幕左上角的位置。

3. Delphi 7 的对象观察器

对象观察器（Object Inspector）又称对象浏览器或属性窗口，是设置窗体和组件属性的工具，使用非常频繁。对象观察器由“对象”列表框、【Properties】选项卡（属性）及事件选项卡【Events】组成，如图 1-6 所示。单击【Properties】可以显示被选中对象的所有属性，单击 Event（事件）选项卡可以显示被选中对象的所有事件。单击某个属性，该属性值（在该属性右边）以选中形式出现，并且可以改变该属性的值。图 1-6 中显示的对象为窗体 Form1，被选中的属性为 Caption，其值为 Form1。

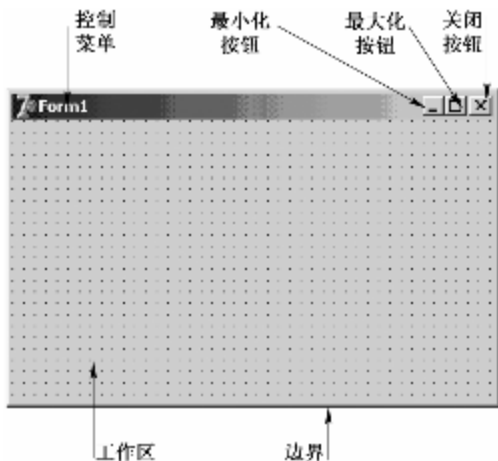


图 1-5 空白窗体

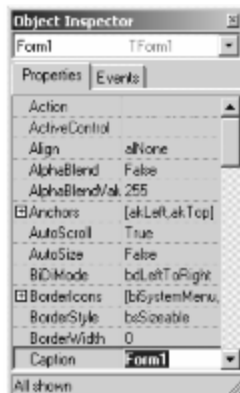


图 1-6 对象观察器

4. 代码编辑器和代码浏览器

代码编辑器和代码浏览器如图 1-7 所示。

代码编辑器是程序代码的输入和编辑工具，也就是用户编写程序源代码的地方。当新建一个窗体后，就自动生成该窗体的单元文件代码，如图 1-7 的右侧所示。




图 1-7 代码编辑器和代码浏览器

启动 Delphi 7 后，屏幕上会出现如图 1-2 所示的集成开发环境窗口，此时初学者可能找不到代码编辑器和代码浏览器，因为它们被窗体遮盖住了。

代码浏览器如图 1-7 中的左侧所示。它列出了单元文件中定义的类型、变量、常量、属性和方法等。代码浏览器支持在代码编辑器中的漫游，即如果双击其中的一个条目，编辑器会自动跳到声明处。代码浏览器和代码编辑器是同时打开的，它们经常被窗体设计器或其他窗口遮盖。

注意：有如下 4 种方法切换到代码编辑器和代码浏览器。

- 1 用鼠标单击没有被窗口遮盖住的代码编辑器和代码浏览器。
- 1 按快捷键 F12。
- 1 单击工具栏上的 Toggle Form/Unit 按钮 .
- 1 单击【View】菜单下的【Code Explorer】选项。

另外，Delphi 7 的 IDE 还有对象树（Object TreeView），如图 1-2 中的左边中间部分。对象树以树状表的形式显示窗体中各种组件之间的所属关系。

1.1.5 利用 Delphi 开发应用程序的方法步骤

现在通过编写一个小程序来介绍利用 Delphi 7 开发应用程序的方法步骤。

【例 1-2】 编写一个显示欢迎词的程序，程序启动时的运行界面如图 1-8 所示。单击【显示】按钮，将在窗体上显示“Delphi 7.0 欢迎您！”，如图 1-9 所示。单击【关闭】按钮，将关闭应用程序。

开发步骤如下。

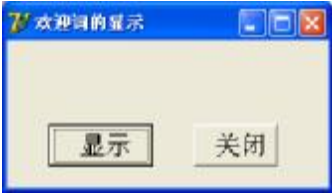


图 1-8 程序运行界面一




图 1-9 程序运行界面二

(1) 新建项目

用户开发的每个 Delphi 程序都叫工程或项目。首先应新建一个项目，启动 Delphi 7 后，自动生成一个项目，项目名为默认的 Project1（在标题栏上），如图 1-2 所示。

注意：新建项目还可以采用以下两种方法。

- 1 单击 File 菜单下 New 子菜单中的 Application 菜单命令（以后这种操作菜单的叙述都统一写成类似于“执行【File】→【New】→【Application】命令”的形式）。
- 1 单击工具栏上的新建按钮, 打开【New Items】对话框, 选中【New】中的【Application】图标，单击【OK】按钮即可。

现在用第一种方法生成一个项目，该项目将自动创建一个名为 Form1 的空白窗体。一个项目可以有多个窗体，第一个窗体默认名为 Form1，第二个默认名为 Form2，其他依此类推。

(2) 保存项目


执行【File】→【Save All】命令或者单击工具栏上的 Save All 按钮保存项目。首先弹出【Save Unit1 As】对话框，如图 1-10 所示。在【保存在】列表框中设置项目文件保存的位置，本例保存位置选择为“D:\DelphiApp\01\A\A_1_2”。在【文件名】文本框中输入要保存的文件名，本例单元文件名采用默认的文件名“Unit1.pas”，设置好后单击【保存】按钮，单元文件保存完毕。此时将出现如图 1-11 所示的【Save Project1 As】对话框，该对话框用来保存项目文件，在【文件名】文本框中输入项目文件名“Wel_Int.dpr”，然后单击【保存】按钮，项目文件保存成功。



图 1-10 【Save Unit1 As】对话框

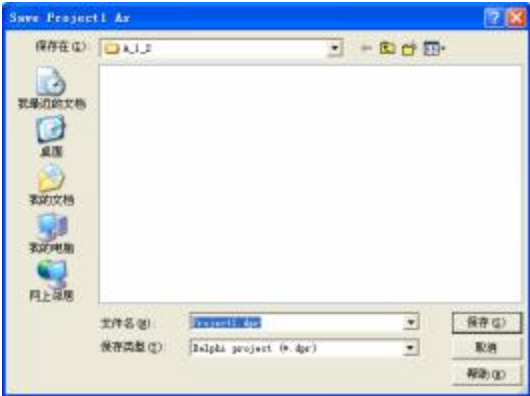


图 1-11 【Save Project1 As】对话框


注意：保存 Delphi 项目文件时，应注意以下几点。

- 1 单元文件名不能和项目文件名及窗体的 Name 属性同名。

- 给项目文件或窗体文件命名应见名思义。一个好的命名可以让用户在一大堆项目或组件中很快明白该项目或组件是干什么的。

(3) 为窗体添加组件

本程序需要在窗体上添加 3 个组件：一个标签组件，用来显示“Delphi 7.0 欢迎您！”，两个命令按钮，分别用来显示欢迎词和退出应用程序。界面设计步骤如下。

① 单击 **Standard** 选项卡中的 **TLabel** 组件（标签组件），在窗体适当位置单击并拖动，**TLabel1** 组件（默认名为 **Label1**）就添加到窗体上了，如图 1-12 所示。


② 单击 **【Standard】** 选项卡中的 **TButton** 组件（按钮），在窗体适当位置单击并拖动就在窗体上添加了一个 **TButton1** 组件（默认名为 **Button1**），用同样的方法添加第二个按钮组件（默认名为 **Button2**），如图 1-13 所示。



图 1-12 添加了 Label1 组件后的窗体

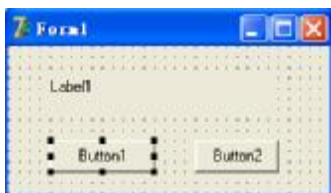


图 1-13 添加了两个 Button 组件后的窗体

注意：删除组件与移动组件位置的方法如下。

- 选中该组件（此时该组件周围出现 6 个小黑点），按 **Delete** 键就可删除该组件。
- 选中该组件，在对象观察器中改变其“**Left**”和“**Top**”属性即可移动该组件。
- 选中该组件，用鼠标拖动至窗体的合适位置也能移动该组件。
- 选中该组件，按住 **Ctrl** 键，同时按下上下左右 4 个方向键可以调整该组件在窗体上的位置。

(4) 设置窗体和组件的属性

① 设置窗体属性。首先单击窗体以选中窗体，在对象观察器中单击 **Caption** 属性名，在其后输入“欢迎词的显示”几个字，用来作为 **Form1** 窗体的标题文本。然后单击 **Font** 属性前的“+”号，将显示 **Font**（字体）属性的所有子属性，单击 **Size**（大小）属性，输入“12”，使窗体上显示的字体较大。

② 设置 **Label1** 属性。单击 **Label1** 以选中该组件，然后单击对象观察器 **【Properties】** 选项卡里的 **AutoSize** 属性，单击右边的下拉箭头，选中“**False**”；单击 **Caption** 属性，在其后输入“Delphi 7.0 欢迎您！”，并拖动标签至适当大小；单击 **Visible** 属性，设置其值为 **False**，使程序刚开始运行时不显示 **Label1** 标签。如图 1-14 所示。

③ 设置 **Button1** 属性。选中 **Button1**，单击 **Caption** 属性，输入“显示”；单击 **Name** 属性，输入“**ButtonDisplay**”。

④ 设置 **Button2** 属性。选中该组件，单击 **Caption** 属性，输入“关闭”；单击 **Name** 属性，输入“**ButtonClose**”。

至此，应用程序的界面已经设置完成，如图 1-15 所示。

需注意的是，每个组件都有 **Name** 属性，大多数组件还有 **Caption** 属性，但它们是有区别的。**Name** 属性与每个组件是一一对应的，是区别每个组件的惟一标志。而 **Caption** 属性只是

该组件的标题属性，用来设置显示的文本。每个组件的 Name 属性不能相同，但 Caption 属性可以相同。



图 1-14 为 Label1 组件设置了 Caption 属性



图 1-15 设计完成后的界面

注意：在对象观察器中和程序运行时设置组件属性有以下 4 种方法。

- ❶ 对文字型属性，如上面按钮的 Caption，直接在该属性右边的文本框输入文字即可。
- ❷ 对数值型属性，如 Left 等位置、大小属性，直接输入数值即可。
- ❸ 对布尔型（只有 True 或 False 两个值）属性和枚举型属性（所有属性都列在该属性文本框里），在属性值列表选择一个即可。
- ❹ 在程序运行时设置某对象（组件）的属性，可使用赋值语句给对象的属性名赋值。

（6）编写程序代码

① 编写【显示】按钮代码。双击该按钮，或选中该按钮，并在【Event】选项卡中 OnClick 事件右边的空白处双击，都将自动生成按钮单击事件的代码框架结构，焦点自动停留在代码编辑器的 begin 和 end 之间，如图 1-16 所示。

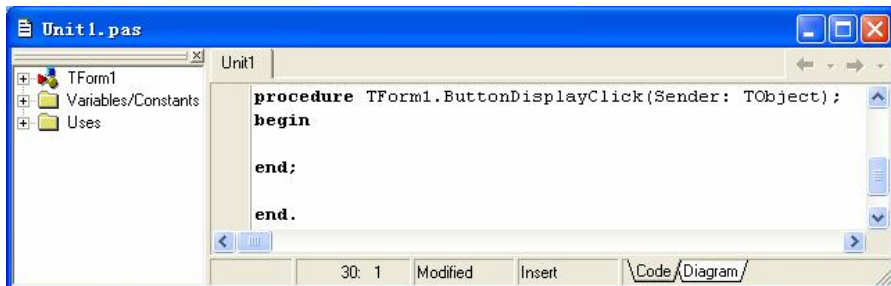


图 1-16 自动产生的事件框架结构

添加如下程序代码：

```
Label1.Visible := True;           //把 Label1 的 Visible 属性设置为 True，以显示该组件
```

② 编写【关闭】按钮代码。用同样的方法编写关闭按钮的代码，代码如下：

```
procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
    Application.Terminate;        //应用程序关闭
end;
```



注意：为了增强程序的可读性，需经常在程序中添加注释。Delphi 有如下 3 种注释方法。

- ❶ 使用双斜杠//。其后的语句就是注释语句，这种注释不能跨行。

! 使用成对的花括号 { }。花括号之间的语句就是注释语句，可以跨行。

! 成对的括号与星号 (* *)。星号之间的语句是注释语句，可以跨行。

(7) 运行程序

程序代码编写完毕后，先保存文件（执行【File】→【Save All】命令或者单击工具栏上的保存按钮），再单击工具栏上的运行按钮或者执行【Run】→【Run】命令即可运行程序。程序运行时，单击【显示】按钮，程序运行结果如图 1-9 所示，单击【关闭】按钮，将结束应用程序的运行。

至此，一个显示欢迎词的应用程序已经创建成功。

1.1.6 Delphi 7 程序的组成

以 Wel_Int.dpr 程序为例，打开该项目所在的目录，发现有 8 个文件，其中只有 3 个文件与程序设计紧密相关：单元文件 Unit1.pas，项目文件 Wel_Int.dpr 和窗体文件 Unit1.dfm，其他文件都是 Delphi 在编译链接时自动产生的，可不必理会。而最重要的是前面两个文件，下面详细说明这两个文件的组成。

1. 单元文件

例 1-2 产生的单元文件 Unit1.pas 的代码如下：

```
unit Unit1;           //unit 是保留字，Unit1 是一个单元名
interface             //接口部分开始
uses                  //以下引用的是单元文件名（库文件）
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm) //定义一个 Form 类
    Label1: TLabel;      //添加标签对象 Label1，作为类的一个成员
    ButtonDisplay: TButton; //添加按钮对象 ButtonDisplay，作为类的一个成员
    ButtonClose: TButton; //添加按钮对象 ButtonClose，作为类的一个成员
    procedure ButtonDisplayClick(Sender: TObject); //ButtonDisplay 的单击事件
    procedure ButtonCloseClick(Sender: TObject); //ButtonClose 的单击事件
  private              //私有变量声明
    { Private declarations }
  public               //公有变量声明
    { Public declarations }
  end;

var
  Form1: TForm1;       //声明一个窗体类对象

implementation         //实现部分开始

{$R *.dfm}             //编译命令
```

```

procedure TForm1.ButtonDisplayClick(Sender: TObject);
begin
    Label1.Visible := True;           //把 Label1 的 Visible 属性设置为 True，以显示该组件
end;
procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
    Application.Terminate;           //应用程序关闭
end;
end.                                 //整个单元代码结束

```

从 Unit1.pas 可见，单元文件包括 3 个部分：标志性语句、接口部分和实现部分。

(1) 标志性语句 “unit”，其后跟随 Unit1，这一句表示 Unit1 是单元文件名。

(2) 接口部分是 interface 和 implementation 之间的部分。不在接口中的条目对于其他单元是不可见的。其中，uses 语句后有很多单元名，说明 Unit1 单元引用了其他单元。一个单元如果要引用其他单元，需要在 uses 后添加其他单元的名字，或者是在 implementation 之后使用 uses 语句添加上其他单元名。保留字 type 与 var 之间是类型说明部分。

(3) implementation 之后是实现部分。它是用户自己输入的程序代码，为完成特定的功能，绝大部分代码都是放在这里。

注意：Delphi 的每条语句都是以分号 “;” 结尾，只有最后一个 end 是以句点 “.” 结尾。

2. 项目文件

项目文件的扩展名为 “.dpr”，由 Delphi 自动生成，一般不必修改。打开项目文件的方法是：执行【View】→【Unit】命令或者按 Ctrl+F12 快捷键，将会显示【View Unit】对话框，选择项目文件即可打开该文件。打开 Wel_Int 文件，该文件代码如下：

```

program Wel_Int;                      //表示这是程序文件，“Wel_Int”为文件名
uses                                  //说明体开始部分，以下是程序引用的单元文件
    Forms,
    Unit1 in 'Unit1.pas' {Form1};
{$R *.res}                           //编译开关，表示要连接的资源文件
begin                                 //程序体开始部分
    Application.Initialize;           //系统初始化工作
    Application.CreateForm(TForm1, Form1); //创建窗体
    Application.Run;                 //程序运行
end.                                 //文件结束

```

从 Wel_Int 代码中可以看到，一个项目文件包括 3 个部分：标志性语句、说明体部分和程序体部分。

(1) 标志性语句 “program”，其后跟随 Wel_Int，表示 Wel_Int 是 Delphi 项目名。

(2) 说明体部分是 uses 与 begin 之间的部分，不包括 begin 语句，用来说明程序引用的单元。本程序引用了两个单元：Forms（Delphi 自带）和 Unit1（用户定义的）。

(3) 程序体部分是 begin 和 end 之间的部分，它是程序实际执行部分。由于大部分代码

都放在单元文件中，故项目文件很小。

1.2 典型实例

【实例题目】：显示与清除文本

编写一个程序完成如下功能：程序运行时，单击【显示】按钮，编辑框（TEdit 组件）显示“你好，中国”，字体大小为 18 号，字体颜色为蓝色；单击【清除】按钮，编辑框显示的文本被清除。程序设计界面如图 1-17 所示，程序运行界面如图 1-18 所示。



图 1-17 程序设计界面





图 1-18 程序运行界面

【实现方法】

欲使 TEdit 组件显示某个文本，如“hello”，只要给 TEdit 组件的 Text 属性赋值为“hello”即可，实现的语句为：`Edit1.Text:='hello';`。另外 TEdit 组件还有字体属性（包括大小，颜色等），按照同样的方法，设置 TEdit 组件的字体颜色为蓝色，字体大小为 18 号即可。为了使 TEdit 组件显示的文本被清除，只要给它的 Text 属性值赋值为空字符串即可。

该程序的实现步骤如下。

- (1) 启动 Delphi 7 新建一个项目。
- (2) 保存项目。单击【File】→【Save All】命令以保存项目，单元文件保存为“Unit1.pas”，项目文件保存为“B_1_1.dpr”。
- (3) 向窗体添加组件。单击【Standard】选项卡中的 TEdit 组件 ，在窗体适当位置单击并拖动鼠标以添加组件 Edit1，再用类似的方法添加按钮组件 Button1 和 Button2。
- (4) 设置窗体和组件属性。根据例 1-2 中提到的方法在对象观察器中设置窗体和组件的属性。选中窗体，单击对象观察器【Properties】选项卡的 Font 属性，设置其 Size 属性值为 12；用鼠标拖动窗体，使其大小合适；选中 Button1，单击对象观察器【Properties】选项卡的 Caption 属性，设置其值为“显示”，单击 Name 属性，设置其值为 ButtonDisplay，单击 Width 属性，设置其值为 60；选中 Button2，用同样的方法设置其 Caption 属性值为“清除”，设置其 Name 属性值为 ButtonClear，设置其 Width 属性值为 60。全部属性设置好以后，再修正窗体大小和组件的相对位置。
- (5) 编写程序代码。
- (6) 保存文件与运行程序。单击【File】→【Save】命令保存文件，再单击工具栏上的运行按钮  或者单击【Run】→【Run】命令以运行程序，程序运行后的界面如图 1-18 所示。

【界面设计】

窗体组件属性设置及其作用如表 1-2 所示。

表 1-2 组件属性及组件功能

组 件 名	属 性 名	属 性 值	作 用
Button1	Caption Name	'显示' ButtonDisplay	单击将显示“你好，中国”
Button2	Caption Name	'清除' ButtonClear	单击将清除 Edit1 中的文本
Edit1	Text	'Edit1'	显示文本
Form1	Width Height	230 172	容纳其他组件等

【程序代码】

```
procedure TForm1.ButtonDisplayClick(Sender: TObject);
begin
    Edit1.Text:='你好，中国';      //设置 Edit1 的 Text 属性为“你好，中国”
    Edit1.Font.Color:=clBlue;      //设置字体颜色为蓝色
    Edit1.Font.Size:=18;           //设置字号为 18 号
end;
procedure TForm1.ButtonClearClick(Sender: TObject);
begin
    Edit1.Text:='';                //清除编辑框中的文本
end;
```

1.3 上机练习

【练习题目】：文本及悬停文字的显示

设置一个文本及悬停文字的显示程序，程序的设计界面如图 1-19 所示。程序运行时，单击【显示】按钮，窗体上显示“开放的中国欢迎你！”文字，如图 1-20 所示。若将鼠标指针停留在【显示】按钮上一小会儿，就显示悬停文字“单击将显示欢迎词”，如图 1-21 所示。单击【关闭】按钮，将关闭应用程序。若将鼠标指针停留在【关闭】按钮上一小会儿也将显示悬停文字“单击将关闭程序”。



图 1-19 程序设计界面



图 1-20 程序运行界面（一）



图 1-21 程序运行界面（二）

【要点提示】

本例题与例 1-2 中编写的程序相似。为了显示文本，可以添加一个标签组件，将欲显示的文本赋值给标签组件的 **Caption** 属性。同时还要添加两个按钮，它们的 **Caption** 属性分别为“显示”和“关闭”。为了显示悬停文字，需要做到两点：在对象查看器中把组件的 **ShowHint** 属性值设置为 **True**，在对象查看器中把组件的 **Hint** 属性值设置为要显示的悬停文本。

【参考代码】

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Caption:='开放的中国欢迎你!';
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    Application.Terminate;      //关闭应用程序
end;
```

注意：利用 Delphi 编写程序时，Delphi 对大小写字母不作区分，但应尽量统一。

课后考场

一、选择题（20 分，每题 5 分）

1. Delphi 7 的单元文件的扩展名是_____。
A. .dpr B. .pas C. .dfm D. .exe
2. Delphi 7 的集成环境不包括_____。
A. 代码编辑器 B. 对象观察器
C. 项目文件优化器 D. 窗体设计器
3. Delphi 7 支持的 3 种注释方法中有一种不能跨行注释，该方法是_____。
A. 使用花括号 B. 使用括号与星号
C. 使用双斜杠 D. 使用星号
4. Delphi 7 语言中，每行可以写多条语句，每条语句（不是最后一条语句）以_____结尾。
A. 逗号 B. 分号 C. 句号 D. 空格

二、填空题（40 分，每空 5 分）

1. Delphi 7 中区别每个组件的惟一属性是_____。
2. 语句 `Button1.ShowHint:=True;` 表示的含义是_____。
3. Delphi 7 中以“.dpr”为扩展名的文件是_____文件。
4. 代码编辑器的作用是_____。
5. 单元文件代码中的最后一条语句是_____。
6. 运行程序可单击工具栏上的_____按钮或按键盘上的_____键。
7. 窗体设计器的主要作用是_____。

三、程序设计题（40 分，每题 20 分）

1. 设计一个显示和隐藏文本的应用程序，程序的设计界面如图 1-22 所示。程序运行时，单击【显示】按钮，文本“国奥冲击奥运”可见，如图 1-23 所示。单击【隐藏】按钮，文本将消失，如图 1-24 所示。请按照可视化程序设计的一般步骤编写程序实现上面的功能。（提示：文本的显示与隐藏，可通过组件的 `Visible` 属性来实现）。



图 1-22 程序设计界面

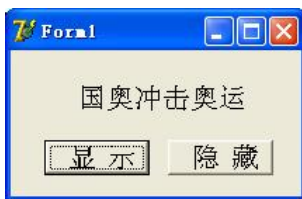


图 1-23 程序运行界面（一）



图 1-24 程序运行界面（二）

2. 设置一个显示悬停文字的应用程序，程序的设计界面如图 1-25 所示。程序运行时，让按钮【打开】显示悬停文字“单击将打开一张图片”，如图 1-26 所示。



图 1-25 程序设计界面



图 1-26 程序运行界面

第 2 章 基本数据类型与表达式

本章要点

- ▮ Delphi 的基本数据类型（整型、实型、字符型、字符串型和布尔型）
- ▮ 常量与变量的概念与定义方法
- ▮ 三类基本运算符：算术运算符、位运算符和字符串运算符的含义与优先顺序
- ▮ 三类基本表达式，并能灵活运用

2.1 理论知识

Delphi 的语法基础是 Object Pascal 语言,因此学习 Delphi 首先应学习并掌握 Object Pascal 语言的语法。本章主要介绍 Object Pascal 语言的语法基础，包括数据类型和表达式。

2.1.1 基本数据类型

程序设计中的数据并不只是数学上的各种数据，而是指描述客观事物的数值、字符和所有能输入到计算机中并能被计算机处理的符号。因此这里的数据比数学上的数据涵盖的范围要广得多。从对数据的定义可以看出，数据有各种各样的类型，类型限定了该数据的表示形式、取值范围和能够参加的各种运算。Delphi 的数据类型如表 2-1 所示，其中前面 5 种是标准数据类型，在使用前不必定义，而后面的高级数据类型在使用前需要用户定义。另外，整型、字符型、布尔型、枚举型和子界型 5 种数据类型为顺序类型，它们的取值是一个有序的非负整数的集合，每个可能的取值都对应一个有序的整数。根据由易到难，循序渐进的学习规律，本章先学习 5 种标准数据类型，分别是：整型、实型、字符型、字符串型和布尔型，其他数据类型将在后面的章节中学习。

表 2-1 Delphi 7 的基本数据类型

类 型	类型说明符	使 用 说 明
整型	Integer	标准数据类型，使用前不必定义
实型	Real	标准数据类型，使用前不必定义
字符型	Character	标准数据类型，使用前不必定义
字符串型	String	标准数据类型，使用前不必定义
布尔型	Boolean	标准数据类型，使用前不必定义
枚举型	Enumerated	高级数据类型，先定义后使用
子界型	Subrange	高级数据类型，先定义后使用

续表

类 型	类型说明符	使 用 说 明
集合类型	Set	高级数据类型, 先定义后使用
数组类型	Array	高级数据类型, 先定义后使用
文件类型	File	高级数据类型, 先定义后使用
记录类型	Record	高级数据类型, 先定义后使用
类类型	Class	高级数据类型, 先定义后使用
类引用类型	Class Reference	高级数据类型, 先定义后使用
接口类型	Interface	高级数据类型, 先定义后使用
指针类型	Pointer	高级数据类型, 先定义后使用
过程类型	Procedural	高级数据类型, 先定义后使用
可变类型	Variant	高级数据类型, 先定义后使用

1. 整型

描述存储各种整数数据的类型即整型。Delphi 7 有 9 种形式的整型数据, 如表 2-2 所示。

表 2-2 Delphi 7 的整型数据

类 型	类型说明符	字 节 数	取 值 范 围
整型	Integer	4	-2 147 483 648~2 147 483 647
序数型	Cardinal	4	0~4 294 967 295
短整型	ShortInt	1	-128~127
长整型	LongInt	4	-2 147 483 648~2 147 483 647
小整型	SmallInt	2	-32 768~32 767
64 位整型	Int64	8	$-2^{63} \sim 2^{63} - 1$
字型	Word	2	0~65 535
长字型	LongWord	4	0~4 294 967 295
字节型	Byte	1	0~255

其中, 最常用的整型为 Integer, 它能满足绝大部分运算要求。在学习之初, 在这么多的数据类型中, 最好记住这 5 种标准数据类型和这 5 种类型中的最常用类型, 其他类型在以后的学习中会较快地记住的。

2. 实型

描述存储各种实数数据的类型为实型。所谓实数就是以小数形式表示的数或以科学记数法表示的数。Delphi 7 有 5 种形式的实型数据, 如表 2-3 所示。其中, Real 型为最常用的实型。

表 2-3 Delphi 7 的实型数据

类 型	类型说明符	字 节 数	取 值 范 围
单精度实型	Single	4	$-3.4 \times 10^{38} \sim -1.5 \times 10^{-39}$, $1.5 \times 10^{-39} \sim 3.4 \times 10^{38}$
双精度实型	Double	8	$-1.7 \times 10^{308} \sim -5.0 \times 10^{-324}$, $5.0 \times 10^{-324} \sim 1.7 \times 10^{308}$
扩展型	Extended	10	$-1.1 \times 10^{4932} \sim -3.6 \times 10^{-4951}$, $3.6 \times 10^{-4951} \sim 1.1 \times 10^{4932}$

续表

类 型	类型说明符	字 节 数	取 值 范 围
货币型	Currency	8	-922 337 203 685 477.580 8~922 337 203 685 477.580 7
实型	Real	8	$-1.7 \times 10^{308} \sim -5.0 \times 10^{-324}$, $5.0 \times 10^{-324} \sim 1.7 \times 10^{308}$

注意：尽量使用 Real 类型而不要使用 Extended 类型，因为 Extended 类型与其他语言或操作平台的兼容性比较差。

3. 字符型

描述存储单个字符数据的类型称为字符型。注意，是存储单个字符。如果要描述存储多个字符，那么就要使用下面将要学到的字符串类型。Delphi 7 有 3 种形式的字符型数据，如表 2-4 所示。

表 2-4 Delphi 7 的字符型数据

类 型	名 称	字 节 数	取 值 范 围
字符型	Char	1(2)	扩展 ANSI 字符集
宽字符型	WideChar	2	Unicode 字符集
Ansi 字符型	AnsiChar	1	扩展 ANSI 字符集

注意：最常用的字符类型为 Char 类型。Char 类型与 Ansi 字符类型完全等价。

4. 字符串型

字符型数据存在一个不足：只能存储一个字符。为存储多个字符，就要使用字符串类型。Delphi 7 有 4 种形式的字符串类型数据，如表 2-5 所示。

表 2-5 Delphi 7 的字符串型数据

类 型	类型说明符	最 大 长 度
字符串型	String	2^{31} 个字符
短字符串型	ShortString	255 个字符
长字符串型	AnsiString	2^{31} 个字符
宽字符串型	WideString	2^{30} 个字符

注意：最常用的数据类型是 String 类型。与字符型数据类似，String 类型与 AnsiString 类型完全等价。

5. 布尔型

布尔型是描述存储 True（真）和 False（假）数据的类型。因此布尔型数据只有两个值 True 和 False。Delphi 7 一共有 4 种形式的布尔型数据，如表 2-6 所示。

表 2-6 Delphi 7 的布尔型数据

类 型	类型说明符	字 节 数	取 值
布尔型	Boolean	1	只能为 0 (False) 或 1 (True)
长布尔型	LongBool	4	0 (False) 或非 0 (True)
宽布尔型	WideBool	2	0 (False) 或非 0 (True)
字节布尔型	ByteBool	1	0 (False) 或非 0 (True)

注意：最常用的类型为 Boolean 类型。

2.1.2 Delphi 7 中的标识符与保留字

与其他语言一样，Delphi 中为了描述一个语法实体，也使用标识符的概念。例如表 2-1 至表 2-6 中各种类型的类型说明符都是标识符，它们是 Delphi 系统规定的。当然也可以自己定义一个符号如“num”，它表示一个整数，“num”就是一个标识符。另外，Delphi 系统规定了很多具有特定意义的单词，在编写代码时不能对它们重新定义或另作它用，这些单词叫保留字，共有 65 个。

1. 标识符

标识符是一种符号，用来表示常量、变量、类型、过程、函数和对象等语法实体。标识符又分标准标识符和自定义标识符。系统规定的标识符叫标准标识符，用户或程序员定义的标识符叫自定义标识符。

(1) 标准标识符

标准标识符包括下面 5 种标识符。

- ┃ 标准常量：例如 Boolean 类型数据的取值 True 和 False。
- ┃ 标准类型：例如表 2-1 至表 2-6 中的各种类型说明名。
- ┃ 标准过程：例如 Put、Reset 等。
- ┃ 标准函数：例如 sin（正弦函数）、cos（余弦）等。
- ┃ 标准文件：例如 Input、TextFile 等。

注意：标准标识符由于是系统预先定义好的，它与下面将介绍的自定义标识符不同，不需要用户定义，可以直接使用。另外，最好不要将标准标识符再定义为自定义标识符，以免混淆。

(2) 自定义标识符

自定义标识符是用户自己定义的描述各种语法实体的名称。自定义标识符的名称由用户根据自己的意愿而定，但必须遵循如下的命名规则：

- ┃ 由字母、数字和下划线“_”组成。
- ┃ 以字母或下划线开头。
- ┃ 不能与保留字同名。
- ┃ 最好不与标准标识符同名。

例如“n_2”、“_n2”均是合法的自定义标识符；而“2n_”和“2_n”均是不合法的标识符；“sin”作为自定义标识符就很不好，容易混淆。

2. 保留字

保留字类似标准标识符，也是系统预先规定好的、具有特定意义的单词。但有一点不同：保留字一定不能重新定义或作为其他用途。Delphi 7 规定了 65 个保留字，如表 2-7 所示。

表 2-7 Delphi 的 65 个保留字

and	array	as	asm	begin	case
class	const	constructor	destructor	dispinterface	div
续表					
do	downto	else	end	except	exports

file	finalization	finally	for	function	goto
if	implementation	in	inherited	initialization	inline
interface	is	label	library	mod	nil
not	object	of	or	out	packed
procedure	program	property	raise	record	repeat
resourcestring	set	shl	shr	string	then
threadvar	to	try	type	unit	until
uses	var	while	with	xor	

注意：public、private、protected、published 和 automated 共 5 个单词在定义类中成员的访问权限时也作为保留字，因此和上面 65 个保留字一样不能重新定义或作其他用途。

3. 指令符

指令符也是具有特定意义的单词，与保留字类似。但它可以作为用户自定义的标识符，这一点又与标准标识符相同。不过为了防止混淆，最好与保留字一样不作其他用途。指令符有 39 个，如表 2-8 所示。

表 2-8 Delphi 的 39 个指令符

absolute	abstract	assembler	automated	cdecl	contains
default	dispid	dynamic	export	external	far
forward	implements	index	message	name	near
nodefault	overload	override	package	pascal	private
protected	public	published	read	readonly	register
reintroduce	requires	resident	safecall	stdcall	stored
virtual	Write	writeonly			

2.1.3 Delphi 7 中的常量与变量

顾名思义，常量是在程序运行过程中其值保持不变的量；变量就是在程序运行过程中其值可以改变的量。

1. 常量

根据使用方式的不同，可以将常量分为直接常量和符号常量。

(1) 直接常量

直接常量就是在程序中直接使用的各种常量，如整型常量 516、-111 等；实型常量 1.2、0.0、3.6E5 等；字符常量 'a'、'10' 等；字符串常量：'www'、'中国' 等；布尔常量 True 和 False。

(2) 符号常量

符号常量就是用一个符号（自定义标识符）来表示常量。符号常量在使用前必须先定义，定义符号常量的定义格式如下。

[格式]：

```
const
    常量名 1=常量值 1;
```

```
常量名 2=常量值 2;  
...  
常量名 n=常量值 n;
```

[功能]: 定义符号常量, 包括符号常量的名字和值。

[说明]: 这里“常量名”与“常量值”之间是“=”符号而不是“:=”。“常量名 1”~“常量名 n”是任意合法的标识符, “常量值 1”~“常量值 n”是直接常量或已经定义的符号常量组成的表达式。

符号常量也可以具有数据类型, 称类型常量。符号常量中的类型可以是标准类型, 也可以是高级类型。类型常量的定义格式如下。

[格式]:

```
const  
    常量名 1:类型名 1=常量值 1;  
    ...  
    常量名 n:类型名 n=常量值 n;
```

[功能]: 定义类型常量, 包括该常量的名字、类型和值。

[说明]: 类型常量的定义与符号常量类似, 仅仅是多了一个类型名。

例如下面的语句:

```
const  
    abc='入门';  
    r:Real=1.414;
```

定义了两个常量, 其一是符号常量, 常量名为 `abc`, 常量值为字符串'入门'; 其二是类型常量, 常量名为 `r`, 类型为实型, 常量值为 1.414。

2. 变量

(1) 变量的声明

变量在使用前必须先声明(或称定义), 声明的位置决定了变量的使用范围。例如, 在函数或过程中声明的变量是局部变量, 只能在该函数或过程中使用; 在单元的 `implementation` 部分声明的变量是单元级(又称模块级、窗体级)变量, 该单元的所有函数与过程都能使用; 在 `interface` 部分定义的变量是全局变量, 其他引用该单元的单元也可以使用该变量。变量具有变量名、变量类型和变量值等特征, 在声明变量时必须声明变量的名称和类型。变量值在程序执行中是可以变化的, 在使用变量之前通常要通过赋值语句给变量赋一个初始值。

声明变量的语法格式及功能如下。

[格式]:

```
var  
    变量名 1:类型名 1;  
    变量名 2:类型名 2;  
    ...  
    变量名 n:类型名 n;
```

[功能]: 声明若干个变量, 包括变量名和变量的类型。

[说明]: 保留字 `var` 表示下面将定义一个或多个变量, “变量名 1” ~ “变量名 n” 是任意合法的标识符, “类型名 1” ~ “类型名 n” 也是任意的标准数据类型或高级数据类型。相同类型的变量可以写在一行, 以逗号分隔。例如:

```
var
  n1,n2,n3:Integer;      //声明 3 个整型变量, 分别为 n1, n2 和 n3
  r1:Real;               //声明 r1 为实型变量
  str:String;            //声明 str 为字符串型变量
```

本例声明了 5 个变量, 每条语句的含义见该语句后面的注释。

(2) 变量的使用

变量经过声明之后就可以使用了。不过变量在使用之前最好先赋初值, 赋初值最简单的办法就是通过赋值语句实现。没赋初值的变量, 其值是个随机数, 在程序运行时, 可能会遇到意想不到的结果。因此变量声明之后最好及时赋初值。

2.1.4 Delphi 7 中的运算符与表达式

Delphi 7 有 8 类运算符, 本章主要学习算术运算符、字符串运算符和位运算符。其他运算符将在以后的章节中学习。

1. 算术运算符

算术运算符是用户最熟悉的, 类似算术中的加减乘除等。Delphi 7 中有 8 个算术运算符, 具体含义如下。

(1) 取正运算符 “+”: 单目运算符, 表示对一个数值型数据取正, 例如 +5 的值为 5。另外, 取正运算符可以省略。

(2) 取负运算符 “-”: 也是单目运算符。例如对 5 取负, 其值为 -5。

(3) 加法运算符 “+”: 与数学上的加法相同。加法运算符与取正运算符符号相同, 但含义不同。例如 3+5 的值为 8。

(4) 减法运算符 “-”: 与数学上的减法含义相同。也请注意减法运算符与取负运算符的异同。

(5) 乘法运算符 “*”: 表示对两个数值型数据相乘, 与数学中的乘法相同。例如: 3*5 的值为 15。

(6) 除法运算符 “/”: 与数学上的除法含义相同。例如 3/5 的值为 “0.6”。另外, 除法运算的结果总是实型数据。

(7) 整除运算符 “div”: 只能对两个整数进行除法运算, 结果为整型数据。例如 5 div 3 的值为 1, 而 “5 div 2.0” 是不合法的。

(8) 取余运算符 “mod”: 对两个整数相除, 结果为余数。例如 5 mod 3 的值为 2。

注意: 关于算术运算符, 应抓住以下几个要点。

■ 取正运算符和取负运算符是单目运算符, 其他 6 个运算符为双目运算符。所谓单目运算符是指仅有一个运算对象的运算符, 双目运算符是指有两个运算对象的运算符。

■ 加法运算符、减法运算符和乘法运算符的结果的类型为参加运算的两个数据中的精度

高的类型。

! /、div 和 mod 运算符中的除数不能为 0。

2. 字符串运算符

Delphi 只有一个字符串运算符“+”——连接运算符，用于把两个或多个字符串连接在一起形成一个新的字符串。例如'abc'+'def'的值为'abcdef'。

3. 位运算符

Delphi 7 有 6 个位运算符，它们的名称和具体含义如表 2-9 所示。

表 2-9 Delphi 7 的位运算符

位 运 算 符	名 称	含 义
Not	按位取反	对操作数的二进制数按位取反
And	按位与	对两个操作数的二进制数按位与
Or	按位或	对两个操作数的二进制数按位或
Xor	按位异或	对两个操作数的二进制数按位异或
Shl	向左移位	对操作数的二进制数按位左移
Shr	向右移位	对操作数的二进制数按位右移

例如：字节型变量 a 和 b，它们的值分别为：00000101 和 00000110，即十进制中的 5 和 6。那么就有以下值。

Not a 的值为：11111010，即十进制中的 250。

a And b 的值为：00000100，即十进制中的 4。

a Or b 的值为：00000111，即十进制中的 7。

a Xor b 的值为：00000011，即十进制中的 3。

a Shl 2 的值为：00010100，即十进制中的 20。

b Shr 2 的值为：00000001，即十进制中的 1。

注意：位运算符具有以下特点。

! 位运算符的操作数必须是整数。

! Not 为单目运算符，其他 5 个运算符为双目运算符。

! 由于 1 个二进制位与 0 相与，结果为 0，与 1 相与结果还是该二进制位，因此按位与运算符通常用来把整数的某些位清 0。

! 由于 1 个二进制位与 1 相或，结果为 1，与 0 相或结果还是该二进制位，因此按位或运算符通常用来把整数的某些位置 1。

! 由于 1 个二进制位与 1 相异或，结果为该二进位的反，与 0 相异或结果还是该二进制位，因此按位异或运算符通常用来把整数的某些位取反。

4. 运算符的优先级

表达式中通常有多个运算符，在计算表达式的值时，必须弄清这些运算符的运算次序，即运算符的优先级。Delphi 7 中的所有运算符的优先级如表 2-10 所示。

表 2-10 Delphi 7 的所有运算符的优先级

运 算 符	优 先 级
-------	-------

Not, @	第一级
*, /, div, mod, and (逻辑与), shl, shr, as (安全类型转换)	第二级
+, -, or (位或), xor (异或)	第三级
=, <, >, <=, >=, in (属于), is (测试对象类型是否匹配)	第四级

注意：级别高的运算符先运算，级别低的运算符后运算。同级运算符，按从左到右的顺序运算。第一级运算符为最高级，第四级为最低级。没有学到的运算符将在以后章节中学习。

5. 表达式

表达式就是用运算符将运算对象（满足该运算符要求的数据）连接起来的式子。在本章中已经接触到了算术表达式、字符串表达式。

（1）算术表达式

算术表达式就是将算术运算符、圆括号和满足该运算符要求的有关数据（数值型常量、变量和函数等）连接起来的式子。算术表达式的结果是一个数值。例如有下面表达式：

$1+2*3-(4/5+(6 \bmod 7))$

该表达式的值为 0.2。

注意：书写算术表达式时，应注意如下几点。

❶ Delphi 语言中的算术表达式与数学意义上的表达式是有区别的，如上面的表达式中没有使用方括号和花括号，在 Delphi 的表达式中只能使用圆括号。

❷ 乘号不能省略。例如 2x 在数学上表示两者相乘，在 Delphi 语句中若表示两者相乘，必须写成 “2*x”。

（2）字符串表达式

利用字符串运算符将字符串常量、字符串变量或字符串函数连接起来的式子称字符串表达式。字符串表达式的格式如下：

$s1+s2$

其中 s1 和 s2 可以是字符串常量、变量或函数。该表达式将 s2 的所有字符连接到 s1 的末尾，并形成一个新的字符串。

例如有如下语句：

`s:='abc'+1234+'56';`

该语句执行后，s 的值为'abc123456'。

2.1.5 Delphi 7 的语句书写规则与赋值语句

1. 语句书写规则

Delphi 的语句分简单语句和复合语句，两者之间并没有太大差别。只是复合语句由两条以上的简单语句组成，同时必须书写在 “begin” 和 “end” 语句之间。两种语句中的每条语句的末尾都是分号（只有程序的最后一条语句是句号）。为区分不同的语句，每条语句以分号隔开，可以在一行写几条语句，也可以把一条语句写在几行上。例如下面的语句是一条简单语句。

`Button1.Enabled:=False;`

该语句的功能是让 Button1 不起作用。语句末尾是一个分号。再看下面的程序段。

```
begin
    t:=a;a:=b;b:=t;
end;
end.
```

其中“begin”和“end;”之间的语句是复合语句，复合语句中还有 3 条简单语句，书写在同一行上，复合语句作为一个整体执行。最后一条语句是“end.”以句号结尾，表示单元程序的结束，“end.”一定是单元的最后一句话。

2. 赋值语句

给变量赋值使用的语句称为赋值语句，赋值语句的语法格式及功能如下。

[格式 1]:

变量名:=表达式;

[功能]: 将表达式的值赋值给变量。

[格式 2]:

对象.属性名:=表达式;

[功能]: 将表达式的值赋值给对象的属性。

[说明]: 表达式的结果与变量或对象的属性同属于一种类型。赋值语句先计算等号右边表达式的值，然后将计算出来的值赋给等号左边的变量或属性。因此赋值语句具有计算和赋值的双重功能。例如如下赋值语句:

```
Label1.Caption:='欢迎您!';    //将'欢迎您!'字符串常量赋值给 Label1 的 Caption 属性
a:=3.5;                        //将实型数据 3.5 赋值给变量 a
```

注意: 赋值号是“:=”而非“=”。

2.1.6 Delphi 7 的常用函数与过程

为了最大限度地减轻用户开发软件的难度，Delphi 系统提供了很多完成通用功能的标准函数与过程。根据函数与过程完成的功能不同，可分为数学类函数、字符类函数、日期时间类函数和顺序类型函数 4 大类。这里介绍前两类函数与过程。

1. 数学类函数

数学类函数包括绝对值函数、取整数函数、取小数函数、平方函数、平方根函数、三角函数、指数函数、对数函数和随机函数等。

(1) 绝对值函数

[调用格式]: Abs(x);

[功能]: 返回 x 的绝对值。

[说明]: 参数 x 是整型或实型数据。

例如:

```
x:=Abs(-1);    //这是一个赋值语句，先计算赋值号右边函数的值（为 1）再赋给 x
```

(2) 取整数函数

[调用格式 1]:

Trunc(x):Int64;

[功能]: 返回实型数据 x 的整数部分, 类型为 **Int64**。

[说明]: 参数 x 是实型数据。

[调用格式 2]:

Round(x):Int64;

[功能]: 返回对实型数据 x 的小数部分四舍五入后的整数部分。

[说明]: 参数 x 是实型数据, 返回的整数是四舍五入后的整数, 与 **Trunc(x)**不同。

[调用格式 3]:

Int(x):Extended;

[功能]: 返回实型数据 x 的整数部分, 类型为 **Extend**。

[说明]: 与 **Trunc(x)**函数基本相同 (返回值类型不同)。

例如:

```
x1:=Trunc(-2.5);           //赋值语句, 函数值-2 赋给 x1
x2:=Round(-2.51);          //x2 的值为-3
x3:=Int(-2.5);              //x3 的值为-2
```

(3) 取小数函数

[调用格式]:

Frac(x):Extended;

[功能]: 返回 x 的小数部分。

[说明]: x 为实型数据。

例如:

```
x:=Frac(-2.5);             //x 的值为-0.5
```

(4) 平方函数

[调用格式]:

Sqr(x):Extended;

[功能]: 返回实型数据 x 的平方值。

[说明]: x 为实型数据。

例如:

```
x:=Sqr(-2.5);              //x 的值为 6.25
```

(5) 平方根函数

[调用格式]:

Sqrt(x):Extended;

[功能]: 返回非负实数 x 的算术平方根。

[说明]: x 为非负实数, 返回值也为非负实数。

例如:

```
x:=Sqrt(4);           //x 的值为 2.0
```

(6) 三角函数

[调用格式 1]:

```
Sin(x):Extended;
```

[功能]: 返回以弧度为单位的 x 的正弦函数值。

[说明]: x 的单位为弧度。

[调用格式 2]:

```
Cos(x):Extended;
```

[功能]: 返回以弧度为单位的 x 的余弦函数值。

[说明]: x 的单位为弧度。

[调用格式 3]:

```
ArcTan(x):Extended;
```

[功能]: 返回 x 的反正切函数值。

[说明]: x 的单位为弧度。

例如:

```
x1:=Sin(1/2);           //x1 的值为 0.4794
x2:=Cos(1/4);           //x2 的值为 0.9689
x3:=ArcTan(1);          //x3 的值为 0.7854
```

(7) 指数函数

[调用格式]:

```
Exp(x):Real;
```

[功能]: 返回值为 e^x 。

[说明]: 参数 x 为实型数据。

例如:

```
x:=Exp(-1);             //x 的值为 0.3679
```

(8) 自然对数函数

[调用格式]:

```
Ln(x):Real;
```

[功能]: 返回 x 的自然对数值。

[说明]: 参数 x 为正实数。其他对数函数可以由换底公式导出。

例如:

```
x:=Ln(5);               //x 的值为 1.6094
```

(9) 随机数函数

[调用格式]:

```
Random(x);
```

[功能]: 返回一个大于等于 0 小于整数 x 的随机整数。

[说明]: x 为整数。若 x 缺省, 返回值为大于等于 0 小于 1 的实数。

2. 字符串处理函数与过程

(1) 大小写转换函数

[调用格式 1]:

```
LowerCase(const s:String):String;
```

[功能]: 把字符串 s 中的字符全部转换为小写字母, 并作为函数值返回。

[调用格式 2]:

```
UpperCase(const s:String):String;
```

[功能]: 把字符串 s 中的字符全部转换为大写字母, 并作为函数值返回。

例如:

```
x1:=LowerCase('aBc');           //将函数值“abc”赋值给字符串变量 x1
x2:=UpperCase('aBc');           //将函数值“ABC”赋值给字符串变量 x2
```

(2) 比较字符串大小函数

[调用格式 1]:

```
CompareStr(const s1,s2:String):Integer;
```

[功能]: 比较两个字符串 s1 和 s2 的大小。

[说明]: 大小比较的依据是 ASCII 码值。根据字符的 ASCII 码值, 从两字符串的第一个字符开始比较, 若前者大于后者, 比较终止, 返回值大于 0; 若小于后者, 比较也终止, 返回值小于 0; 若等于后者, 将接着继续比较后面的字符。直到遇到不相等的字符或 s1 或 s2 的所有字符均比较完毕。若所有位置上的字符都相等, 则返回值等于 0。注意: 本函数区分大小写。

[调用格式 2]:

```
CompareText(const s1,s2:String):Integer;
```

[功能]: 比较字符串大小。

[说明]: 比较方法同上, 区别是本函数不区分大小写。

例如:

```
x1:=CompareStr('Abc','abc');      //x1 的值为-32
x2:=CompareText('Abc','abc');     //x2 的值为 0
```

(3) 求字符串长度函数

[调用格式]:

```
Length(s):Integer;
```

[功能]: 返回字符串的长度。

[说明]: 参数 s 的类型为字符串型。

例如:

```
x:=Length('abcd');               //x 的值为 4
```

(4) 查找位置函数

[调用格式]:

```
Pos(s1,s2:String):Integer;
```

[功能]: 返回子字符串 s1 在字符串 s2 中首次出现的位置, 若 s2 中不存在 s1, 则返回 0。

[说明]: 若 s1 在 s2 中, 则必须满足 s1 的所有字符都在 s2 中, 即 s1 是 s2 的子串。

例如:

```
x1:=Pos('bc', 'abcd');      //x1 的值为 2
x2:=Pos('bd', 'abcd');      //x2 的值为 0
```

(5) 合并字符串过程

[调用格式]:

```
AppendStr(var s1:String;Const s2:String);
```

[功能]: 相当于执行语句 “s1:=s1+s2;”。

[说明]: 本过程比语句 “s1:=s1+s2;” 执行效率高。

(6) 截取子字符串函数

[调用格式]:

```
Copy(s:String;m,n:Integer):String;
```

[功能]: 在字符串 s 中截取从第 m 个字符开始长度为 n 的子字符串, 并作为函数的返回值。

[说明]: 若 m 大于 s 的长度, 则返回一个空串; 若从第 m 个字符到 s 的结尾不足 n 个字符, 则返回其间的所有字符。

例如:

```
s1:=Copy('abcdef',2,2);      //s1 的值为'bc'
s2:=Copy('abcdef',4,5);      //s2 的值为'def'
```

注意: Copy('abc',0,1)和 Copy('abc',1,1)的函数值相等, 都是'a'。

(7) 删除子字符串过程

[调用格式]:

```
Delete(var s:String;m,n:Integer);
```

[功能]: 在字符串 s 中删除从第 m 个字符开始长度为 n 的子字符串。

[说明]: 如果 m 大于 s 的长度, 则不删除任何字符; 如果从第 m 个字符开始到 s 的末尾不足 n 个字符, 则删除其间的所有字符; 如果 n 小于等于 0, 则不删除任何字符。

例如下面的程序段:

```
var
  s:String;                      //字符串变量声明
begin
  s:='abcd';                    //给变量赋值
  Delete('abcd',3,3); {删除从第 3 个字符开始长度为 3 的子字符串 (本例只能删除 'cd')}
  Edit1.Text:=s;                //s 的值为子串'ab', 故 Edit1 上显示的文本为'ab'
end;
```

(8) 插入子字符串过程

[调用格式]:

```
Insert(S1:String;var s:String;k:Integer);
```

[功能]: 将字符串 S1 插入到字符串 s 中的第 k 个字符处。

(9) 数值和字符串相互转换函数

[调用格式 1]:

`IntToStr(m:Integer):String;`

[功能]: 将整型数据 m 转换成字符串并作为函数的返回值。

[调用格式 2]:

`StrToInt(s:String):Integer;`

[功能]: 将字符串数据 s 转换成整型数据并作为函数的函数值。

[说明]: s 为数字型字符串。

[调用格式 3]:

`FloatToStr(f:Extended):String;`

[功能]: 将实型数据 f 转换成字符串, 并为函数的返回值。

[调用格式 4]:

`StrToFloat(s:String):Extended;`

[功能]: 将字符串型数据 s 转换成实型数据, 并作为函数的返回值。

[说明]: s 为数字型字符串。

2.2 典型实例

【实例题目】: 求整数各位数字

利用算术运算符和算术表达式编写一个程序, 实现下面的功能: 输入一个任意 4 位整数, 计算千位、百位、十位和个位上的数字。程序设计界面如图 2-1 所示, 程序运行界面如图 2-2 所示。程序运行时, 输入任意整数, 单击**【个位数字】**按钮, 将显示个位数字, 单击**【十位数字】**按钮, 将显示十位数字, 依此类推。



图 2-1 程序设计界面



图 2-2 程序运行界面

【实现方法】

对于一个 4 位整数, 要知晓如何求其各位数字。例如, 对于 1234 这样一个整数, 将它整除 1000 就得到千位上的数值 1; 将 1234 对 1000 取余得到 234, 再整除 100 即是百位上的数字 2; 将 1234 对 100 求余得到 34, 再对 34 整除 10 就得到十位上的数字 3; 1234 对 10 取余即得到个位上的数字 4。

【界面设计】

窗体及组件的属性设置如表 2-11 所示。

表 2-11 组件属性设置及组件作用

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'输入任意整数'	提示在右边编辑框输入任意整数
Label2~Label5	Caption	"	显示各位数字
Edit1	Text	"	用来输入任意整数
Button1~Button4	Caption	'个位数字'~'千位数字'	单击显示相应位数字

【程序代码】

(1) 双击 Button4，编写求千位上的数字的程序代码：

```
procedure TForm1.Button4Click(Sender: TObject);
var
    t:Integer;                      //声明 t 为整型变量
begin
    t:=StrToInt(Edit1.Text);        //字符串转换成整型数据再赋值给 t
    Label5.Caption:=IntToStr(t div 1000); //div 表示整数除法，求千位上的数字
end;
```

(2) 双击 Button3~Button1，编写求百位、十位和个位上数字的代码：

```
procedure TForm1.Button3Click(Sender: TObject);
var
    t:Integer;
begin
    t:=StrToInt(Edit1.Text);
    Label4.Caption:=IntToStr((t mod 1000) div 100); //求百位上的数字
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    t:Integer;
begin
    t:=StrToInt(Edit1.Text);
    Label3.Caption:=IntToStr((t mod 100) div 10); //求十位上的数字
end;
procedure TForm1.Button1Click(Sender: TObject);
var
    t:Integer;
begin
    t:=StrToInt(Edit1.Text);
    Label2.Caption:=IntToStr(t mod 10); //求个位上的数字
end;
```

2.3 上机练习

【练习题目】：简单的函数计算器

常用的计算器能求很多种函数的值，用户也可以编写一个程序计算某些函数的值。程序的设计界面如图 2-3 所示，程序运行时输入自变量 x ，单击【Sin(x)】、【Tan(x)】、【Sqr(x)】和【Lg(x)】按钮，将分别计算出 x 的相应函数值，并显示在函数值后面的编辑框中。程序运行界面如图 2-4 所示。



图 2-3 程序设计界面



图 2-4 程序运行界面

【要点提示】

对于正弦、余弦和正切函数，其自变量是以弧度为单位的。常用对数函数没有现成的标准函数，但可以通过换底公式实现，换底公式为： $\text{Lg}(x) = \text{Ln}(x)/\text{Ln}(10)$ 。

【参考代码】

```
implementation
var
    x:Real;           //声明一个单元级变量（注意声明位置），下面的所有函数与过程都能引用
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    x:=StrToFloat(Edit1.Text);           //字符串转换成实型数据
    Edit2.Text:=FloatToStr(Sin(x));       //实型数据转换成字符串
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    x:=StrToFloat(Edit1.Text);
    Edit2.Text:=FloatToStr(Sin(x)/Cos(x)); //求正切函数
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    x:=StrToFloat(Edit1.Text);
    Edit2.Text:=FloatToStr(Sqr(x));       //求平方函数
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    x:=StrToFloat(Edit1.Text);
```

```
Edit2.Text:=FloatToStr(Ln(x)/Ln(10)); //利用换底公式求常用对数
end;
```

课后考场

一、选择题（20分，每题5分）

- 下列自定义标识符不合法的是_____。
A. i*m B. im C. i_m D. mi
- 下列标识符中_____不是保留字。
A. And B. Near C. END D. Mod
- 算术表达式 $10+20 \bmod (\text{Trunc}(5.5))*2$ 的值是_____。
A. 19 B. 14 C. 10 D. 23
- 数学表达式 $\sin 60^\circ$ 的 Delphi 表达式是_____。
A. $\sin(60)$ B. $\sin(60*3.14159/180)$
C. $\text{SIN}(60)$ D. $\sin(60^\circ)$

二、填空题（40分，每空5分）

- 在 Delphi 语言中，自定义标识符第一个字符只能是_____。
- 函数 $\text{CompareStr}('abcd', 'abCD')$ 的值_____零。
- 函数 $\text{Length}('ab+cd')$ 的值是_____。
- 在下面程序段的划线处填空：

```
var
    _____
begin
    s:='Chinese';
    Delete(s,4,5);
end;
```

执行后 s 等于_____。

- 表达式 $(30+22 \bmod 4) \div 8$ 的值是_____。
- 表达式 $\frac{-b - \sqrt{b^2 - 4ac}}{2a}$ 的 Delphi 语言表达式是_____。
- 表达式 $\text{UpperCase}('Hello') + '+' + \text{LowerCase}('China')$ 的值是_____。

三、程序设计题（40分，每题20分）

1. 请编写程序实现下面的功能：程序设计界面如图 2-5 所示，程序运行时，单击【随机数 1】按钮，产生一个 100 以内的非负整数；单击【随机数 2】同样产生一个 100 以内的非负整数。单击【加法运算】按钮，将把两个随机数做加法运算，结果将显示在【答案】的编辑框里；单击【乘法运算】按钮，将把乘法运算的结果显示在【答案】编辑框里。程序运行界面如图 2-6 所示。



图 2-5 程序设计界面



图 2-6 程序运行界面

2. 编程计算 10^x 的值。程序设计界面如图 2-7 所示，程序运行界面如图 2-8 所示。



图 2-7 程序设计界面



图 2-8 程序运行界面

第3章 基本程序设计语句

本章要点

- ▮ 顺序结构、选择结构和循环结构程序设计的思想
- ▮ 双分支语句和多分支语句的含义
- ▮ 利用双分支语句和多分支语句设计分支程序的方法
- ▮ 循环的概念
- ▮ 循环程序设计的方法及常用的算法

3.1 理论知识

结构化程序设计有三种基本结构，即顺序结构、选择结构和循环结构。任何复杂的程序均可分解成这三种基本结构，这三种基本结构也能够组合成任意复杂的程序。

3.1.1 基本的顺序结构语句及其应用

顺序结构是程序设计中最常用也是最简单的程序结构。所谓顺序结构就是其中的语句是按照先后顺序依次执行，程序执行的流程不会发生跳转。其实，不管采用哪种结构，顺序结构都贯穿其中，例如，选择结构和循环结构中的语句，在大多数情况下都是按顺序执行的。顺序结构语句主要是由赋值语句等简单的操作语句组成。赋值语句的格式与功能已在第2章作了介绍。

【例3-1】 设计一个程序，程序设计界面如图3-1所示。在程序设计时，编辑框显示“远方的朋友，欢迎您”文字，字号为12号，字体为宋体，字体颜色为红色。程序运行时，单击【改变】按钮，编辑框文本内容不变，但字号改为20号，字体变为隶书，颜色变为蓝色，如图3-2所示。



图 3-1 程序设计界面



图 3-2 程序运行界面

【实现分析】

要在程序运行时改变编辑框文本的有关属性，只需通过赋值语句给相应属性赋值就可实

现。如本例中，使文本字号大小变为 20 号的语句如下：

```
Edit1.Font.Size:=20;
```

其他属性也采用同样的赋值方法，不再赘述。

【界面设计】

本例的组件属性设置及组件功能如表 3-1 所示（在对象观察器中设置）。

表 3-1 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'改 变'	单击将改变编辑框字体颜色字号字体名
Edit1	Text Font.Size Font.Name Font.Color	'远方的朋友，欢迎您' 12 宋体 clRed	显示文本内容

【程序代码】

双击【改变】按钮，编写如下程序代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Font.Name:='隶书';           //设置编辑框文本字体名为隶书
    Edit1.Font.Size:=20;               //设置编辑框文本字体大小为 20 号
    Edit1.Font.Color:=clBlue;         //设置编辑框文本颜色为蓝色
end;
```

3.1.2 选择结构语句及其基本应用

日常生活中经常要根据一个条件是否满足来决定是否做某件事。对于这类问题，在程序设计中可通过选择结构来完成。所谓选择结构，就是对给定的条件进行分析判断，在满足不同的条件时执行不同的操作。选择结构可以分成单分支选择结构、双分支选择结构和多分支选择结构。

1. 关系表达式与逻辑表达式

要掌握选择结构，首先要掌握条件的表示方法——关系表达式和逻辑表达式。

(1) 关系运算符和关系表达式

关系运算符就是用来对两个表达式进行大小比较的运算符。用关系运算符把两个运算对象连接起来表示它们大小关系的式子称为关系表达式。如果关系表达式描述的关系成立，则关系表达式的结果为 True，否则结果为 False。关系运算符一共有 9 种，其符号及名称如表 3-2 所示。

表 3-2 关系运算符及名称

运 算 符	名 称	运 算 符	名 称	运 算 符	名 称
=	等于	>=	大于等于	<=	包含于
<>	不等于	<	小于	>=	包含
>	大于	<=	小于等于	in	属于

例如, $5 \geq 3$ 是一个关系表达式, 通过大于等于运算符 “ \geq ” 把两个表达式 5 和 3 连接起来了, 该表达式的值为 **True** (真)。又如 $'a' > 'z'$ 也是一个关系表达式, 但其值为 **False**。因为 $'a'$ 的 ASCII 码值为 97, 而 $'z'$ 的 ASCII 码值为 122, 显然前者小于后者。

关系表达式的一般格式及其功能如下。

[格式]:

表达式 1 关系运算符 表达式 2

[功能]: 比较两个表达式值的大小。

注意: 关系运算遵循以下规律。

- 1 关系表达式两边的数据类型必须一致。
- 1 9 个关系运算符的运算级别相同。在同级别运算符中, 按从左到右的顺序运算, 括号里的表达式最先运算。
- 1 当对字符进行大小比较时, Delphi 对字母的大小写是有区别的, 并按照每个字符的 ASCII 码值大小予以比较。常见字符的 ASCII 码值按从小到大的顺序是: 空格 $< '0' < \dots < '9' < 'A' < \dots < 'Z' < 'a' < \dots < 'z' < \text{任何汉字}$ 。

(2) 逻辑运算符和逻辑表达式

如果天晴又有时间, 我就去打球, 否则就不去。这是一个选择结构的例子, 对于选择结构, 可以暂时不理睬。先看看前面的条件: 天晴又有时间。很显然, 这里有两个条件, 都必须满足, 才会去打球。对于这种复合条件, 使用关系运算符已经无法描述, 必须使用逻辑运算符和逻辑表达式来描述。逻辑运算符一共有 4 个, 它们的名称及有关说明见表 3-3。

表 3-3 逻辑运算符及其含义

运 算 符	名 称	含 义
Not	逻辑非	单目运算符。进行取“反”操作, 由 True 变 False 或由 False 变 True
And	逻辑与	双目运算符。只有两个表达式的值同时为 True, 结果才为 True, 否则结果为 False
Or	逻辑或	双目运算符。只要有一个表达式的值为 True, 结果就为 True, 否则结果为 False
Xor	逻辑异或	双目运算符。当两个表达式的值不同则结果为 True, 否则为 False

逻辑表达式就是用逻辑运算符把若干个关系表达式或逻辑值 (True 或 False) 连接起来的式子。逻辑表达式的值与关系表达式的值一样, 也只有两个: True 或 False。例如: $('a' \geq 'c') \text{Or False}$ 就是一个逻辑表达式。其中 $('a' \geq 'c')$ 是一个关系表达式, 而 False 是一个逻辑值, Or 是“逻辑或”运算符, 上式的值为 False。

逻辑表达式的运算步骤是: 对双目运算符, 先运算两个表达式的值, 再进行逻辑运算; 对单目运算符 (Not), 先计算表达式的值, 再进行逻辑非操作。再来分析刚才的例子 $('a' \geq 'c') \text{Or False}$, 双目逻辑运算符 Or 左边是一个关系表达式 $('a' \geq 'c')$, 右边是一个逻辑值 False, 关系表达式的值为 False, 逻辑值也是 False, 因此根据表 3-3 的说明, 整个逻辑表达式的值为 False。

注意: 逻辑运算遵循以下规律。

- 1 逻辑运算符与第 2 章讲解的位运算符符号相同, 但含义不同。当逻辑运算符两边的数据为数值时, 此时的逻辑运算符转化为位运算符, 结果为整数而不是逻辑值。
- 1 逻辑运算符的优先级别是: Not 为第一级, And 为第二级, Or 和 Xor 为第三级。先运算级别高的, 后运算级别低的, 同级别运算符, 按从左到右的顺序运算。而括号里的

表达式最先运算，不管其级别高低。

2. 单分支与双分支 IF 语句

IF 语句常用来实现单分支或双分支，它的语法格式有以下两种。

[格式 1]:

```
IF (条件) Then
    语句 1;
```

[功能]: 如果条件为 **True** 就执行语句 1。

[说明]: 如果条件为 **False**，就不执行语句 1，因此又称为单分支语句。格式中的条件可以是关系表达式或逻辑表达式。

[格式 2]:

```
IF (条件) Then
    语句 1
Else
    语句 2;           //Else 之前（语句 1 之后）没有分号
```

[功能]: 如果条件为 **True** 就执行语句 1，否则执行语句 2。

[说明]: 不管条件是否为 **True**，都要执行一个分支语句，因此又称为双分支语句。格式中的条件可以是关系表达式或逻辑表达式。

注意:

(1) 对语法格式 2 (双分支 IF 语句)，语句 1 后面没有分号，因为 IF...Then...Else 语句是一个完整的语句。

(2) 两种语法格式中的语句 1 和语句 2 都可以是简单语句或复合语句 (复合语句必须包含在 **begin** 和 **end** 之间)。

例如:

```
IF (t1>t2) Then
    t1:=5.5;
```

这是一个单分支条件 (单分支 IF) 语句。当 (t1>t2) 的值为 **True** 时，t1 被赋值为 5.5。而下面的语句为双分支条件语句 (双分支 IF 语句)。

```
IF (t1>t2) Then
    t1:=5.5
Else
    t1:=0;
```

当 (t1>t2) 的值为 **True** 时，t1 被赋值为 5.5，否则 t1 被赋值为 0。

当格式中的语句 1 或语句 2 又是一个 IF 语句时，则称为 IF 语句的嵌套。例如以下程序:

```
IF (a>=0) Then
begin
    IF (a>0) Then
        a:=b+c
    Else
        //Else 前没有分号
//复合语句中嵌套 IF 语句
```



```

a:=b-c;           //表示 a=0 时, a 被赋值为 b-c
end
Else
a:=0;             //表示 a<0 时, a 被赋值为 0

```

【例 3-2】 编写一个做算术题的程序，程序的设计界面如图 3-3 所示。程序执行时，单击【出题】按钮，将随机产生 100 以内的非负加数和被加数，当在等号“=”右边的方框里写出正确答案后，单击【结果】按钮，将显示“答案正确”；如果输入的答案不正确，将显示“答案错误”，如图 3-4 所示。

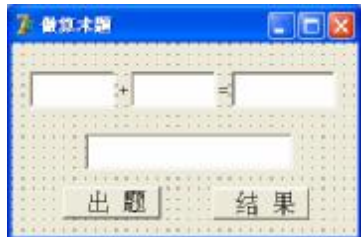


图 3-3 程序设计界面



图 3-4 程序运行界面

【实现分析】

为了产生 100 以内的非负整数，可以调用随机函数 `Random`。例如 `Random(100)` 即可产生 100 以内的非负整数。答案正确显示一种文本，答案错误显示另一种文本，因此本例是双分支 IF 语句，可以用 `IF...Then...Else` 格式的语句来实现。

【界面设计】

本例的组件属性设置及组件功能如表 3-4 所示（在对象观察器中设置）。

表 3-4 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'出 题'	单击将随机产生 100 以内的非负整数
Button2	Caption	'结 果'	单击显示有关文本
Label1	Caption	'+'	提示做加法运算
Label2	Caption	'='	提示在右边编辑框写出答案
Edit1	Text	"	用来显示被加数
Edit2	Text	"	用来显示加数
Edit3	Text	"	用来填写答案
Edit4	Text	"	显示有关文本

【程序代码】

```

procedure TForm1.Button1Click(Sender: TObject);
var
    num1,num2:Integer;           //定义 2 个局部整型变量
begin
    Edit3.Text:= "";            //清空编辑框

```

```
Edit4.Text:= "";
num1:=Random(100);           //将随机产生的小于 100 的非负整数赋值给 num1
num2:=Random(100);
Edit1.Text:=IntToStr(num1);   //将整型数据转换成字符串赋值给编辑框
Edit2.Text:=IntToStr(num2);
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    n1,n2,n3:Integer;         //定义 3 个局部整型变量
begin
    n1:=StrToInt(Edit1.Text);  //将字符串转换成整型数据赋值给变量
    n2:=StrToInt(Edit2.Text);
    n3:=StrToInt(Edit3.Text);
    if (n3=n1+n2) then         //双分支 IF 语句
        Edit4.Text:='答案正确' //条件为真, 执行本语句
    else
        Edit4.Text:='答案错误'; //条件为假, 执行本语句
    end;
end;
```

3. 多分支 Case 语句

当分支很多时, 用 IF 语句来实现, 需要嵌套很多层, 写书不但烦琐, 而且也容易出错。为实现多分支, Delphi 提供了 Case 语句, 该语句又称多分支语句, 其语法格式与功能如下。

[格式]:

```
Case (表达式) of
    值 1:语句 1;
    值 2:语句 2;
    ...
    值 n:语句 n;
Else                               //这里的 Else 之前可以有分号
    语句 n+1;
end;
```

[功能]: 根据变量的值, 决定执行某个分支语句。

[说明]: 执行 Case 语句时, 首先计算“表达式”的值, 如果是“值 1”到“值 n”之间的某个值, 则执行该值后面对应的语句, 否则执行 Else 后面的语句, 执行一个分支后将跳转到 end;后面的语句执行。

注意:

(1) “表达式”的值必须是顺序类型, 即整型、字符型、布尔型、子界型和枚举型。

(2) “值 1”到“值 n”应该是“表达式”可能出现的值, 它们还应该各不相同。如果“表达式”的某几个值对应的执行语句是相同的, 那么可以将这几个值写在同一行上, 以逗号分隔。例如, 当变量等于值 1 和值 2 时, 都执行语句 1, 那么可以这样写:

```
Case (表达式) of
    值 1,值 2:语句 1;
```

…（下面的语句与上面格式相同，略）

- （3）语句可以是简单语句，也可以是复合语句。
- （4）Else 语句可以省略，省略时如果“表达式”的值与“值 1”～“值 n”中的任何一个值均不相等，则不执行任何分支，直接跳转到 end 后面的语句执行。
- （5）Case 语句可以用 IF 语句来实现，但 IF 语句不一定能用 Case 语句实现。

【例 3-3】 某个团体按如下规定交会费：收入在 0～499 元之间的交 0.5%，收入在 500～999 元之间的交 1%，收入在 1000～1999 元之间的交 1.5%，收入在 2000 元及以上的交 2%。编写程序实现：根据输入的收入金额，计算出应该上交的会费。程序设计界面如图 3-5 所示，程序运行界面如图 3-6 所示。



图 3-5 程序设计界面



图 3-6 程序运行界面

【实现分析】

本例含有多种情况，用多分支语句 Case 来实现比较方便。根据 Case 语句的要求，表达式必须是顺序类型，因此本例必须根据收入构造出一个顺序型的表达式，如整型表达式。通过分析可知，本例中上交会费的转折点发生在 500 的整数倍处，为简化起见，可以这样做：假定收入为 t，首先计算 k 的值，计算公式为：k:=(t div 500)；当 k 的值为 0 时，上交会费的比例是 0.5%；当 k 等于 1 时，上交会费的比例是 1%，其他以此类推。因此可用 k 作为 Case 语句中的表达式。

【界面设计】

本例的组件属性设置及组件功能如表 3-5 所示。

表 3-5 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'计 算'	单击将计算应交的会费
Label1	Caption	'收入='	提示输入收入金额
Label2	Caption	'会费='	提示这是应该上交的会费
Edit1	Text	"	显示收入金额
Edit2	ReadOnly	True	显示上交的会费

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    f:Real;           //定义变量
    t,k:Integer;
```

```
begin
    t:=StrToInt(Edit1.Text);           //将字符串转换成整型数据赋值给 t
    k:=t div 500;                       //整除
    Case k of                           //多分支 Case 语句
        0:f:=t*0.5/100;                //当收入在 0~499 元时, 交 0.5% 的会费
        1:f:=t*1/100;
        2,3:f:=t*1.5/100;
    else
        f:=t*2/100;
    end;
    Edit2.Text:=FloatToStr(f);         //将实型数据转换成字符串赋值给 Edit2.Text
end;
```

3.1.3 循环结构语句及其基本应用

顺序结构和选择结构只能完成一些简单功能, 实际的程序也很少只由顺序结构和选择结构组成。用计算机解决许多问题都必须通过循环结构, 可以说没有循环结构就没有程序设计。从程序设计的角度来看, “循环”是指某一个程序段重复执行若干次, 被重复执行的程序段称“循环体”。在程序中为控制循环的执行, 通常需要设定一个条件, 当该条件成立时执行循环, 当条件不成立时, 退出循环。把这样的控制循环是否执行的条件称“循环条件”或“循环控制条件”。

Delphi 7 实现循环结构的语句一共有 3 种, 分别是 While 语句、Repeat 语句和 For 语句。

1. While 语句

While 语句就是当条件成立 (为 True) 时, 执行循环, 而在条件不成立 (为 False) 时, 将退出循环, 不再执行循环体。While 语句的语法格式及功能如下。

[格式]:

```
While (循环条件) do    //循环条件为关系或逻辑表达式, 其值为 True 或 False
    循环体;             // “循环体”可以是简单语句或复合语句
```

[功能]: 当循环条件为 True 时, 执行循环体。

[说明]: 执行 While 语句时, 首先计算“循环条件”, 若其值为 True, 则执行循环体中的语句, 否则不执行循环体中的语句, 跳出循环 (或者称终止循环)。每执行完一次循环体中的语句后, 都要重新计算“循环条件”, 若其值为 True, 则接着执行循环体中的语句, 否则不执行循环体中的语句, 跳出循环。

注意:

- (1) 可以在循环体的任何位置放置 Break 语句, 用来强制终止整个循环。
- (2) 也可以在循环体的任何位置放置 Continue 语句, 在该次循环执行完之前就结束本次循环, 重新判断循环条件, 根据循环条件的值, 决定是否继续执行循环。显然, Continue 语句和 Break 语句含义是不同的。
- (3) Break 语句和 Continue 语句通常放置在循环体中的 IF 语句之后, 即在满足某个条件的时候, 结束循环或结束本次循环。
- (4) 注意循环条件, 它决定了是否执行循环以及执行多少次循环。

【例 3-4】 利用 While 语句求 2+3+...+50 的和。程序设计界面如图 3-7 所示，程序运行界面如图 3-8 所示。



图 3-7 程序设计界面



图 3-8 程序运行界面

【实现分析】

求 2+3+...+50 的和就是不断地重复做两个数的加法，因此可以用循环语句来实现。对于循环语句，一定要保证循环条件的正确，因为循环条件决定了循环的次数。本例中的循环条件就是加数是否小于或等于 50。假设累加和为 s，加数为 k，那么使用表达式 “s:=s+k;” 即可实现加法，k 的初值应为 2，同时每做一次加法，还应该使 k 加 1，即 k:=k+1;。

【界面设计】

本例的组件属性设置及组件功能如表 3-6 所示。

表 3-6 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'计 算'	单击它将求累加和
Label1	Caption	'2+3+...+50='	提示这是求累加和
Edit1	ReadOnly	True	显示累加结果

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  k,s:Integer;           //定义整型变量
begin
  s:=0;                  //给变量赋初值
  k:=2;
  While (k<=50) do       //While 循环语句
  begin                  //循环体为复合语句
    s:=s+k;              //实现累加
    k:=k+1;              //每加一次，循环变量加 1
  end;
  Edit1.Text:=IntToStr(s);
end;
```

2. Repeat 语句

Repeat 语句用来实现“直到型循环”，其语法格式及功能如下。

[格式]:

Repeat

循环体;

Until (循环条件);

[说明]: 执行 Repeat 语句时, 首先执行循环体, 执行完后, 再判断循环条件, 如果循环条件为 False (注意不是 True), 继续执行循环体。直到某一次判断循环条件, 循环条件为 True, 此时终止循环。

对照 While 语句的执行过程, 不难看出 Repeat 语句和 While 语句的区别: Repeat 是先执行循环体, 后判断条件, 因此循环体至少要执行一次; While 是先判断条件, 只有在条件为 True 时, 才执行循环体, 因此有可能循环体一次也不执行。

注意:

(1) 循环体可以是简单语句, 也可以是复合语句, 对于复合语句, 不需要用 begin...end 括起来。

(2) 也可以使用 Break 和 Continue 语句来终止循环或结束本次循环。

(3) 在“循环条件”为 False 时执行循环, 为 True 时退出循环。

【例 3-5】 用 Repeat 语句实现例 3-4 的功能, 程序设计界面和程序运行界面分别如图 3-7 和图 3-8 所示。

【实现分析】

Repeat 语句是直到型循环语句, 即当条件为 True 时结束循环, 因此循环条件与 While 语句不同。由于累加求和是从 2 加到 50, 因此当加数大于 50 时应该停止加法运算, 故本例中的“循环条件”应该是大于 50。

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  k,s:Integer;           //定义整型变量
begin                   //给变量赋初值
  s:=0; k:=2;
  Repeat                //Repeat 循环语句
    s:=s+k;             //实现累加
    k:=k+1;             //每加一次, 循环变量加 1
  Until (k>50);          //当加数大于 50 就中断循环, 循环条件与【例 3-4】不同
  Edit1.Text:=IntToStr(s);
end;
```

上面程序段执行完毕后, s 的值为 1274。这里请思考: 能否将循环条件改为“k=50”? 如果一定要这样改, 那么其他语句该如何修改?

3. For 语句

在循环次数未知的情况下, 使用上述的两条语句很容易求解。在循环次数已知的情况下, 使用 For 循环语句将会使编程更加方便。For 循环语句的语法格式及功能如下。

[格式]:

For 循环变量:=初值 To (DownTo) 终值 Do
循环体;

[功能]: 执行规定次数的循环。

[说明]: 执行 For 循环语句时, 对递增循环而言 (使用 “To”), 首先把初值赋给循环变量, 然后判断循环变量的值是否大于终值, 如果是, 将结束循环的执行; 如果不是, 将执行循环体, 循环体执行后使循环变量自动增加 1, 并开始新的循环, 即通过判断循环变量与终值的关系以决定是执行循环体还是结束循环。对递减循环而言 (使用 “DownTo”), 也是首先把初值赋给循环变量, 然后判断循环变量的值是否小于终值, 如果是, 将结束循环的执行; 如果不是, 将执行循环体, 循环体执行后使循环变量自动减少 1, 并开始新的循环, 即通过判断循环变量与终值的关系以决定是执行循环体还是结束循环。

注意:

- (1) “循环变量” 只能是顺序类型 (一共 5 种类型, 请查阅前面章节)。
- (2) 格式中的 “初值” 和 “终值” 就是循环变量的 “初值” 和 “终值”。
- (3) 对递增循环, 每循环一次, 循环变量的值自动增加 1; 对递减循环, 每循环一次, 循环变量自动减少 1。
- (4) 循环体可以是简单语句也可以是复合语句, 若为复合语句, 需要用 begin...end 括起来。
- (5) 在循环体中可以使用 Continue 和 Break 语句, 它们也通常位于 IF 语句之后。

【例 3-6】 用 For 语句实现例 3-4 的功能, 程序设计界面和程序运行界面仍然如图 3-7 和图 3-8 所示。

【实现分析】

对于 For 语句, 必须准确地知道循环变量的初值和终值, 本例中, 循环变量是加数, 其初值应该是 2, 终值是 50。

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    k,s:Integer;           //定义整型变量
begin
    s:=0;                  //给变量赋初值
    for k:=2 to 50 do      //注意循环条件及其初值和终值
        s:=s+k;           //实现累加
    Edit1.Text:=IntToStr(s);
end;
```

可见, 对于次数已知的循环, For 语句比 While 语句和 Repeat 语句简练多了。

3.2 典型实例

3.2.1 典型实例一

【实例题目】: 邮箱登录程序

在很多场合需要输入密码，例如使用银行卡、登录电子邮箱、QQ 聊天等。下面利用 IF 语句编写一个类似登录服务的简单程序，程序的设计界面如图 3-9 所示。程序实现的功能如下：在编辑框中输入密码，单击【确定】按钮，如果输入的密码正确（假定正确密码是“123abc”），就在另一个编辑框显示“欢迎使用本系统”文本信息，文本的字号为 18 号，字体为隶书，字体颜色为蓝色，如图 3-10 所示。如果输入的密码不正确，就显示“警告，密码错！”文本信息，文本字号还是 18 号，字体为楷体，字体颜色为红色，如图 3-11 所示。



图 3-9 程序设计界面



图 3-10 程序运行界面（一）

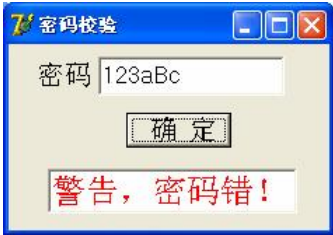


图 3-11 程序运行界面（二）

【实现方法】

本例显然是一个双分支结构：当密码正确，显示一个文本；当密码错误，显示另一个文本。因此需要用 IF...Then...Else 语句来实现。

【界面设计】

窗体组件属性设置及其作用如表 3-7 所示。

表 3-7 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'确定'	根据密码正确与否单击将显示不同文本
Edit1	Text	"	用来输入密码
Edit2	Text ReadOnly	" True	根据输入的密码正确与否显示相关内容

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if (Edit1.Text='123abc') then           //Edit1.Text='123abc'为关系表达式
    begin                                   //当关系表达式值为真时执行下面的复合语句
        Edit2.Text:='欢迎使用本系统';      //文本内容为欢迎使用本系统
        Edit2.Font.Name:='隶书';           //文本字体为隶书
        Edit2.Font.Color:=clBlue;          //字体颜色为蓝色
        Edit2.Font.Size:=18;               //字号为 18 号
    end
    else                                   //else 之前没有分号
    begin                                   //当密码错执行下面的复合语句
        Edit2.Text:='警告，密码错！';      //文本内容为“警告，密码错！”
        Edit2.Font.Name:='楷体';           //字体变为楷体
    end
end
```



```
    Edit2.Font.Size:=18;
    Edit2.Font.Color:=clRed;           //字体颜色为红色
end;
end;
```

3.2.2 典型实例二

【实例题目】：购物打折程序

节日来临，商家为促销搞优惠活动，具体优惠办法如下：

- (1) 一次性购物金额少于 100 元的，不优惠；
- (2) 一次性购物金额大于等于 100 元但少于 300 元的，优惠 5%；
- (3) 一次性购物金额大于等于 300 元但少于 500 元的，优惠 10%；
- (4) 一次性购物金额大于等于 500 元但少于 1000 元的，优惠 15%；
- (5) 一次性购物金额大于等于 1000 元的，优惠 20%。

请编写程序实现上面的功能，当输入购物金额时，单击【付款】按钮，计算实际支付的金额。程序设计界面如图 3-12 所示，程序运行界面如图 3-13 所示。



图 3-12 程序设计界面



图 3-13 程序运行界面

【实现方法】

本例是一个多分支选择结构，可使用 Case 语句来实现。使用 Case 语句实现多分支的关键是构造 Case 后面的表达式。通过分析可知，由于优惠办法规定的转折点的金额刚好是 100 的整数倍，因此可用购物金额取整数部分再整除 100 作为 Case 后面的表达式。若表达式的值为不同的值时，执行的语句是一样的，则可把它们合并到一个分支中。例如当购物金额少于 500 元但大于等于 300 元时，优惠比例是相同的，该语句可以这样写：3,4: s:=t*(1-0.1);其中，s 表示实际支付的金额，t 表示购物金额。

【界面设计】

窗体组件属性设置及其作用如表 3-8 所示。

表 3-8 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'付款'	单击将计算出实际应该支付的购物金额
Edit1	Text	"	用来输入购物金额
Edit2	Text ReadOnly	" True	显示实际应该支付的购物金额

续表

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'购物金额'	提示输入购物金额
Label2	Caption	'实际支付'	提示实际支付的购物金额

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    m:Integer;
    t,s:Real;
begin
    t:=StrToFloat(Edit1.Text);
    m:=Trunc(t);
    Case (m div 100) of
        0: s:=t;
        1,2: s:=t*(1-0.05);
        3,4: s:=t*(1-0.1);
        5,6,7,8,9: s:=t*(1-0.15);
    else
        s:=t*(1-0.2);
    end;
    Edit2.Text:=FloatToStr(s);
end;
```

3.2.3 典型实例三

【实例题目】：计算数学中常量 e 的值

编一个程序按下列公式求 e 的值（要求精度达到 10⁻⁵）。

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\dots+\frac{1}{n!}$$

程序的设计界面如图 3-14 所示，程序运行时，单击【计算】按钮，将计算并显示符合给定精度的 e 值，如图 3-15 所示。

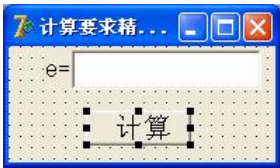


图 3-14 程序设计界面

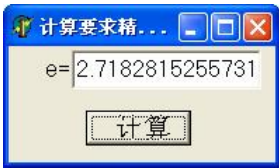


图 3-15 程序运行界面

【实现方法】

本例是一种循环求解连加的典型题型。对于这类题，可把它想像成这样的一个债主收债

的模型：债主放一个盒子 SUM；若干个欠钱的人排成队，给欠钱的人按序编一个号 I（项的号），第一个人的号为 1；每个欠钱的人手里拿着要还的钱 T（该项的值），准备把钱放到盒子里去；一开始盒子里无钱，故 SUM 的值为 0，第一个人准备还钱，I 的值为 1；然后算出该人手中拿着的钱 T，该人把钱放到盒子中，此时盒子里的钱为原来盒子里的钱加上刚放进去的这个人手上拿的钱；该人离开，下一个人准备还钱，序号 I 的值应加 1；再算出第 I 个人手上拿着的钱；把钱放到盒子中；盒子里的钱为前面 I 个人手上拿的钱的和；序号 I 的值再加 1……如此循环，直到所有欠钱的人都把手中的钱放到盒子里，此时盒子里的钱就是应收的总账（总和）。

对于连加求和的式子关键要解决两个问题：一是如何求加到的那一项值 T，二是如何控制循环的条件。求 T 的方法有两类：一是项的值 T 是项数 I 的通式；二是通过迭代来求下一项，即后一项值 T 可以通过前一项来得到。通过分析可以得到这样的规律：如果把第二项看成第一项（ $1/1!$ ），那么第 I 项的值就是第 I-1 项的值除以 I，即后一项可以由前一项来得到。本题循环条件的控制采用了具体条件法，即当某一项的值 T 小于 10^{-5} 时退出循环。

【界面设计】

窗体组件属性设置及其作用如表 3-9 所示。

表 3-9 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Form1	Caption	'计算要求精度的 e 值'	容纳其他组件
Button1	Caption	'计算'	单击将计算出符合精度要求的 e 值
Label1	Caption	'e='	提示这是符合精度要求的 e 值
Edit1	Text ReadOnly	" True	用来显示满足精度要求的 e 值

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    i:Integer;           //定义变量
    s,t:Real;
begin
    i:=1;                //i 代表项号，1/1!为第一项
    s:=1;                //s 代表和，初值为 1
    t:=1;                //t 代表要加到的那一项的值，初值为 1
    Repeat
        t:=t*1.0/i;      //求每个加数即一般项
        s:=s+t;           //累加求和
        i:=i+1;           //每做一次加法，循环变量增加 1
    Until (t<=0.00001);  //符合要求的精度即终止循环
    Edit1.Text:=FloatToStr(s);
end;
```

3.2.4 典型实例四

【实例题目】：素数判断程序

编写一个应用程序，判断某数是否为素数，程序的设计界面如图 3-16 所示。程序运行时，在编辑框中输入任意一个正整数，单击【判断】按钮，在另一个编辑框显示该数是否为素数，如图 3-17 所示。要求：通过 For 语句来实现。

【实现方法】

首先要知道素数的求法。假定求整数 k 是否为素数，有三种方法。方法一：用 2 到 k-1 中的每个数去除 k，只要有一个余数为零，说明 k 不是素数；方法二：用 2 到 k/2（取整数）中的每个数去除 k，只要有一个余数为零，说明 k 不是素数；方法三：用 2 到 Sqrt(k)（取整数）中的每个数去除 k，只要有一个余数为零，说明 k 不是素数。本例采用方法二。



图 3-16 程序设计界面

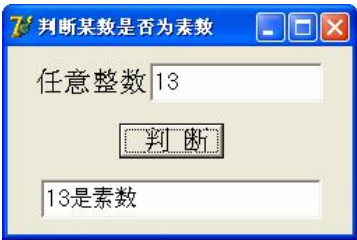


图 3-17 程序运行界面

【界面设计】

窗体组件属性设置及其作用如表 3-10 所示。

表 3-10 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'判断'	单击将判断是否为素数
Label1	Caption	'任意整数'	提示输入任意整数
Edit1	Text	"	用来输入任意整数
Edit2	Text ReadOnly	" True	显示该整数是否为素数

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    m,n,k,j:Integer;
begin
    m:=StrToInt(Edit1.Text);
    k:=m div 2;                //整除
    j:=0;
    for n:=2 to k do
    begin
```

```

if ((m mod n)=0) then           //循环结构中又包含选择结构
begin
    j:=1;                       //能够整除，说明不是素数
    break;                      //发现不是素数，立即终止整个循环
end;
end;
if (j=1) then
    Edit2.Text:=Edit1.Text+'不是素数'    // “+” 是字符连接符
else
    Edit2.Text:=Edit1.Text+'是素数';
end;

```

3.2.5 典型实例五

【实例题目】：枚举法求百钱买百鸡问题

我国古代有个著名的百钱买百鸡问题：用 100 元钱买 100 只鸡，公鸡每只 5 元，母鸡每只 3 元，小鸡每 3 只 1 元，问该如何买？买法有许多种，请找出其中的一种。程序设计界面如图 3-18 所示，程序运行界面如图 3-19 所示。



图 3-18 程序设计界面

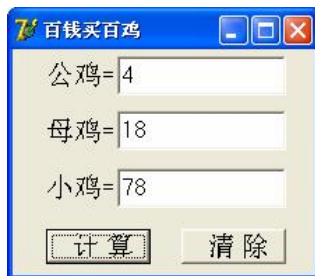


图 3-19 程序运行界面

【实现方法】

对这类问题，首先可以列写数学方程求解。假设买公鸡 m 只，母鸡 n 只，小鸡 k 只，根据题意，可以得到如下的方程组：

$$\begin{cases} m + n + k = 100 \\ 5m + 3n + k/3 = 100 \end{cases}$$

显然这是一个三元一次方程组，但只有两个方程，无法直接求解，可以用枚举法一个一个来试：即将所有的可能拿来一个一个地试验，看哪些满足要求，满足要求的即是要寻找的答案。对于本例，可买的公鸡数应为 1~19、可买的母鸡数应为 1~33，小鸡数应为 100 减去公鸡数与母鸡数，并且小鸡数应能被 3 整除。对于每一个情况看是否刚好花完了 100 元钱，若是则是要求的一个解，输出公鸡数、母鸡数和小鸡数。

【界面设计】

窗体组件属性设置及其作用如表 3-11 所示。

表 3-11 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'计 算'	单击将找到一种买法（显示公鸡母鸡和小鸡数）
Button2	Caption	'清 除'	单击清空编辑框
Label1	Caption	'公鸡='	提示右边为公鸡数
Label2	Caption	'母鸡='	提示右边为母鸡数
Label3	Caption	'小鸡='	提示右边为小鸡数
Edit1	Text	"	用来显示公鸡数
Edit2	Text	"	用来显示母鸡数
Edit3	Text	"	用来显示小鸡数

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    nc,nh,nch,flag:integer;           //定义 4 个整型变量，flag 用来标记是否找到
begin
    flag:=0;                          //给 flag 赋初值 0
    for nc:=1 to 19 do                 //公鸡的取值范围为 1 至 19
    begin
        for nh:=1 to 33 do            //母鸡的取值范围为 1 至 33，for 语句嵌套 for 语句
        begin
            nch:=100-nc-nh;           //小鸡的取值为 100 减去公鸡和母鸡数
            //当买鸡的钱刚好等于 100 时，且有小鸡能够被 3 整除，此时是一个解
            if (((nch mod 3)=0) and ((nc*5+nh*3+nch/3)=100)) then
            begin
                Edit1.Text:=IntToStr(nc);    Edit2.Text:=IntToStr(nh);
                Edit3.Text:=IntToStr(nch);
                flag:=1;                    //已经找到
                break;                      //找到一种答案即可，因此用 Break 语句终止所有循环
            end;
        end;
    end;
    if (flag=1) then                  //如果已经找到，退出外层循环
        break;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Edit1.Text:="";                  //清空三个编辑框
    Edit2.Text:=""; Edit3.Text:="";
end;
```

3.3 上机练习

3.3.1 上机练习一

【练习题目】：求阶乘

关于阶乘的问题数学上也经常遇到，请编写程序求 $n!$ 的值。程序设计界面如图 3-20 所示，程序运行界面如图 3-21 所示。



图 3-20 程序设计界面

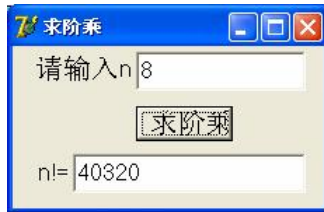


图 3-21 程序运行界面

【要点提示】

阶乘是典型的连乘问题，与连加可采用同样的分析方法，只是运算符号为*号，乘积的初值应为 1。

【参考代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    t,n,s:Integer;
begin
    t:=1;                                //t 表示乘到的那一项的值，赋初值 1
    n:=StrToInt(Edit1.Text);            //n 代表自然数，要求它的阶乘
    s:=1;                                //存放乘积，即阶乘的值
    Repeat
        s:=s*t;                          //连乘
        t:=t+1;                          //每循环一次，变量增加 1
    Until (t>n);                          {注意循环条件，能否写成 m>=n? 如果能，程序该如何修改? 请思考}

    Edit2.Text:=IntToStr(s);
end;
```

3.3.2 上机练习二

【练习题目】：求两数之间不能被 4 整除的数

在两个编辑框中任意输入两个正整数，求它们之间所有不能被4整除的数，并显示出来。程序设计界面如图3-22所示，程序运行界面如图3-23所示。请编程实现上面的功能。



图3-22 程序设计界面

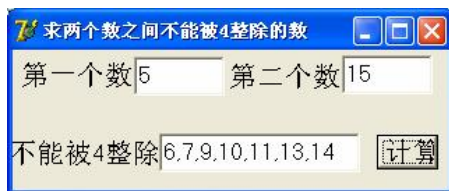


图3-23 程序运行界面

【要点提示】

可通过 IF 语句来判断某数是否能够被4整除，使用的关系表达式为“ $K \bmod 4 = 0$ ”。通过一个循环依次判断两个数之间的每一个数是否能够被4整除，如果能被4整除就结束本次循环，重新开始下一次循环，以便判断下一个数是否能被4整除。注意：结束本次循环应使用 Continue 语句而不是 Break 语句。

【参考代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
    m,n,k,j:Integer;           //定义整型变量
    str:String;                 //定义字符串变量
begin
    m:=StrToInt(Edit1.Text);    n:=StrToInt(Edit2.Text);
    str:="";
    if (m>n) then
    begin                       //如果 m>n，就交换它们的值
        k:=m;    m:=n;    n:=k;
    end;
    for j:=m+1 to n-1 do
    begin
        if (j mod 4)=0 then
            continue;           //如果能被4整除，就终止本次循环，开始新的循环
        str:=str+IntToStr(j)+','; //如果不能被4整除，就添加到字符串中
    end;
    Edit3.Text:=Copy(str,1,Length(str)-1); //去掉最后一个分号
end;
```

课后考场

一、选择题（20分，每题5分）

1. 结构化程序设计的三种基本结构是 _____。
A. 顺序结构、选择结构和分支结构

- B. 选择结构、分支结构和循环结构
C. 分支结构、循环结构和重复结构
D. 顺序结构、分支结构和循环结构
2. 以下关系式语法正确的是_____。
A. $99 > '89'$ B. $2 < m \leq 7$ C. $a > B$ D. $'a' > 'B'$
3. Case 语句格式中“表达式”的类型可以是_____。
A. 字符串型 B. 任何类型 C. 实型 D. 字符型
4. 下列的 Button1Click 事件执行后, Edit1 中显示的文本为_____。

```
procedure TForm1.Button1Click(Sender: TObject);
var
  a,b:integer;
begin
  b:=1;
  for a:=1 to 100 do
  begin
    if (b>=10) then break;
    if (b mod 3=1) then
    begin
      b:=b+3; continue;
    end
  end;
  Edit1.Text :=inttostr(a);
end;
```

- A. 101 B. 6 C. 5 D. 4

二、填空题（40分，每空5分）

1. 表达式 $(3 \geq 3) \text{And} (2 * 5 > 7)$ 值是_____。
2. x 是不大于 50 的非负数的逻辑表达式是_____。
3. 判断闰年的逻辑表达式是_____。（闰年是指能够被 4 整除不能被 100 整除的年，或者是能够被 400 整除的年。）
4. 循环语句根据循环条件和循环变量以决定是否执行循环和执行多少次循环，但 Repeat 语句不管循环条件如何，都至少要执行_____次循环体。
5. 有下面的程序段：

```
procedure TForm1.Button1Click(Sender: TObject);
Var
  a,b,c:integer;
begin
  a:=2;b:=3;c:=1;
  Edit1.text:='End';
  if (a>b) then
```

```

if (a>c) then
    Edit1.Text:=inttostr(a)
else
    Edit1.Text:=inttostr(b);
end;

```

则程序执行后，在 Edit1 组件中显示的文本为_____。

6. 假设有如下程序段：

```

i:=1.0;
While i<=3 do
    i:=i+0.5;

```

请问循环体的执行次数是_____。

7. Case 语句中的条件表达式的值必须是_____类型。

8. 分析如下程序段：

```

var  n:Longint;
n:=1234567;
for k:=1 to 7 do
    n:=n mod 10;
n:=n div 10;

```

则程序段执行后，n 的值是_____。

三、程序设计题（40 分，每题 20 分）

1. 为了安全起见，可以将邮箱登录程序稍作修改，实现如下的功能：当密码输入正确（假定正确密码是 123abc），单击【确定】按钮，显示“欢迎使用本系统”；如果输入的密码错误，单击【确定】按钮，显示“密码错，请注意大小写”；如果连续三次输入错误密码，显示红色文本“对不起，你无权使用”，并且密码输入框和命令按钮不可用。程序设计界面如图 3-24 所示，程序运行界面如图 3-25 所示。

2. 同构数是指这样的整数：它正好出现在其平方数的右端，例如 5 的平方数是 25，5 出现在其平方数的右端，因此 5 是同构数。编写程序找出 1~100 以内的全部同构数。程序设计界面如图 3-26 所示，程序运行界面如图 3-27 所示。



图 3-24 程序设计界面



图 3-25 程序运行界面



图 3-26 程序设计界面



图 3-27 程序运行界面

第 4 章 数组程序设计

本章要点

- ▮ 数组的概念
 - ▮ 一维数组和二维数组的含义与定义方法
 - ▮ 引用数组元素的方法
 - ▮ 与数组有关的常用算法
-

4.1 理论知识

当需要对很多相同类型的数据进行处理时，如果使用变量，将是非常烦琐而且容易出错。如要处理一个班 60 个人的数学成绩，若使用变量就必须定义 60 个变量来存放成绩，显然非常麻烦。为解决上述问题，在 Delphi 中引进了数组类型。数组适合把一组具有相同类型的数据组合在一起，并使用相同的名字——数组名。数组中的每一个成员称数组元素，并可以通过可变的下标来访问数组的每一个元素，从而为程序处理一组相同性质的数据带来了方便。

4.1.1 数组的概念

数组是一些具有相同类型的元素按一定顺序组成的序列。其中每个元素由其对应的位置来指定，这个位置就是数组的索引号（又称下标），数组元素与索引号是一一对应的，用户可以通过这个索引号来存取数组的每个元素。数组中的各元素是顺序地安排在内存中一段连续的存储空间中。按照数组在定义时是否确定了元素个数可以将数组分为静态数组和动态数组。静态数组在定义时就确定了元素的个数和元素的数据类型，并且不可改变。动态数组则没有确定数组元素个数，在程序运行过程中根据需要动态地决定数组的元素个数。数组按照其维数也常分为一维数组、二维数组和 multidimensional 数组，一维数组只有一个下标，二维数组有两个下标，多维数组较复杂，本章不作讨论。本章只讨论一维数组和二维数组。

4.1.2 一维静态数组的定义与使用

1. 一维静态数组类型的定义

一维静态数组类型的定义格式与功能如下。

[格式]:

Type

数组类型名=array[下标类型] of 基类型

[功能]: 定义一个一维静态数组类型, 类型名由“数组类型名”指定。

[说明]: 数组类型名为任意合法的标识符; 保留字 `array` 表示定义数组类型; 下标类型为整型、字符型、布尔型、子界型和枚举类型等; 基类型为任意类型; 当下标类型和基类型为高级类型时, 需要在使用前声明; 等号“=”不要写成赋值号“:=”。

例如, 下面的语句定义一个一维数组类型 `num`:

```
Type  
    num=Array[1..10]of Integer;
```

由于程序不能直接使用类型, 因此定义(声明)了数组类型后, 还必须定义(声明)数组变量, 例如下列语句:

```
var  
    num1:num;
```

这样就声明了一个 `num` 类型的数组变量 `num1`。`num1` 的各元素及其排列位置如图 4-1。其中, `num1[1]` 为数组变量 `num1` 的第一个元素, `num1[2]` 为数组变量 `num1` 的第二个元素, 以此类推。

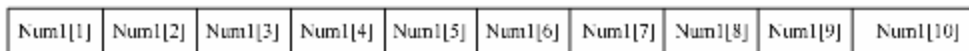


图 4-1 数组 `num1` 各元素排列位置示意图

上面采用的方法是先定义数组类型后定义数组变量, 其实也可以在定义数组类型的同时定义数组变量, 两种方法是完全等效的。例如:

```
var  
    num1:Array[1..10]of Integer;
```

2. 一维静态数组元素的使用

程序中一般并不整体地使用数组, 而是使用数组元素, 引用数组元素的格式及功能如下。

[格式]:

数组名[下标]

[功能]: 引用数组的某个元素。

[说明]: 不要把格式中的方括号“[]”写成其他括号。

例如, 有下列语句:

```
for k:=1 to 10 do  
    num1[k]:=k;
```

该循环给数组中的每个元素赋了一个值, 赋值后 `num1[1] ~ num1[10]` 的值分别是 1~10。

一维数组元素相当于一个同类型的变量, 凡是能够使用变量的地方均可以使用同种类型的数组元素。

【例 4-1】 随机产生 5 个两位正整数并存放到一维数组中, 然后找出其中的最大值及其下标。程序设计界面如图 4-2 所示, 程序运行界面如图 4-3 所示。



图 4-2 程序设计界面

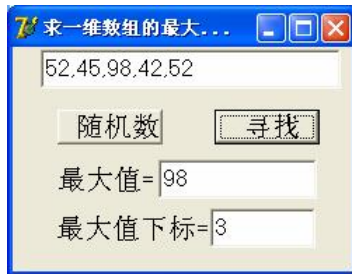


图 4-3 程序运行界面

【实现分析】

寻找最大值及其下标的方法是：将 5 个元素进行编号，根据数组的概念把 5 个元素赋值给一个数组的各元素自然就编好号了。假定数组的最大元素值为 \max ，最大元素的下标为 n_{\max} ，首先将第一个元素值及其下标赋值给 \max 和 n_{\max} 。然后将 \max 与数组第二个元素比较，若 \max 小于第二个元素，则将第二个元素值赋值给 \max ，下标赋值给 n_{\max} ，否则不赋值；再将 \max 与第三个第四个直至最后一个元素比较， \max 小则 \max 和 n_{\max} 被赋值，否则不赋值。逐一比较完毕，就找到了最大元素及其下标。

为了随机产生两位数的正整数可以调用 `Random` 函数。例如 `Trunc(10*Random)` 将随机产生 0~9 的整数。由于 `Random` 函数产生的数为实型数据，因此调用了 `Trunc` 取整函数。

由于数组变量在产生随机数和寻找最大值的过程中都要用到，数组变量应在 `Implementation` 部分定义，详见程序代码。

【界面设计】

本例的组件属性设置及组件功能如表 4-1 所示。

表 4-1 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'最大值='	提示右边为最大值
Label2	Caption	'最大值下标='	提示右边为最大值的下标
Edit1	ReadOnly	True	容纳随机产生的 5 个两位正整数
Edit2	ReadOnly	True	显示最大值
Edit3	ReadOnly	True	显示最大值下标
Button1	Caption	'随机数'	单击产生 5 个随机正整数
Button2	Caption	'寻找'	单击找出最大值及其下标

【程序代码】

```
implementation
var
    RanNum:Array[1..5]of Integer;           //定义单元级数组变量，下面的函数与过程都可调用
    {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    m:Integer;                               //这是局部变量，只能在本函数或过程里使用
```

```

begin
    Randomize;           //为产生互不相同的随机数，程序启动时调用一次 Randomize 过程
    Edit1.Text:="";      //清空编辑框
    Edit2.Text:=""; Edit3.Text:="";
    for m:=1 to 5 do
    begin
        RanNum[m]:=Trunc(10+90*Random);           //随机产生 10~99 的整数
        Edit1.Text:=Edit1.Text+IntToStr(RanNum[m])+','; //将随机数赋给编辑框
    end;
    Edit1.Text:=Copy(Edit1.Text,1,Length(Edit1.Text)-1); //去掉最后一个逗号
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    max,n_max,m:Integer;           //这是局部变量，因此 m 可以同名
begin
    max:=RanNum[1];
    n_max:=1;
    for m:=2 to 5 do
        if (max<RanNum[m]) then
        begin
            max:=RanNum[m];           //如果 max 小于数组的某个元素，就将该元素值及其
            n_max:=m;                 //下标赋给 max 和 n_max
        end;
    Edit2.Text:=IntToStr(max);
    Edit3.Text:=IntToStr(n_max);
end;

```

4.1.3 二维数组及多维数组的定义与使用

1. 二维静态数组类型的定义

二维静态数组比一维静态数组多一个下标，其定义格式及功能如下。

[格式 1]:

Type

数组类型名=Array[下标类型 1, 下标类型 2] of 基类型

[格式 2]:

Type

数组类型名=Array[下标类型 1] of Array [下标类型 2] of 基类型

[功能]: 定义二维数组类型，类型名由“数组类型名”指定。二维数组有两个下标。

多维数组类型的定义和使用与二维数组类似，定义格式与功能如下。

[格式]:

Type

数组类型名=Array[下标类型 1, 下标类型 2, ..., 下标类型 n] of 基类型

[功能]: 定义多维数组类型。

[说明]: N 维数组类型有 N 个下标。下标类型与基类型含义同前。一般很少使用三维以上的数组。

二维静态数组类型定义后,还要声明二维数组变量,声明的方式与一维静态数组基本相同,此处不再赘述。

同样,二维数组元素也相当于一个同类型的变量,凡是能够使用变量的地方均可以使用同种类型的数组元素。

2. 二维静态数组的使用

在使用二维数组之前必须先定义二维数组类型和声明二维数组变量。一般也不整体地使用二维数组,而是使用二维数组的数组元素。引用二维数组的数组元素的格式与功能如下。

[格式]:

二维数组名[下标 1,下标 2]

[功能]: 引用二维数组的数组元素,数组元素位置由“下标 1”和“下标 2”指定。

[说明]: 两个下标都必须写出来,而且下标 1 和下标 2 不能混淆。

【例 4-2】 编写程序实现下面的功能:输入两个学生三门功课的成绩,并统计每个学生的总成绩。程序设计界面如图 4-4 所示,程序运行界面如图 4-5 所示。



图 4-4 程序设计界面



图 4-5 程序运行界面

【实现分析】

有两个学生,每个学生有 4 个分数(每门课程和总分),相当于一个 2 行 4 列的矩阵,因此声明一个二维数组来接受学生的成绩。由于该数组在输入成绩和统计总成绩时都要用到,因此其作用域应该是单元级的,可在 Implementation 部分定义。

重复地输入学生的每门课程的成绩可以利用循环语句来实现。为输入各门课程成绩,可使用 InputBox 函数,注意该函数返回值类型为字符串。InputBox 函数的格式与功能如下。

[格式]:

变量=InputBox(对话框标题,信息内容,默认值)

[功能]: 弹出输入对话框,允许用户输入数据,并把用户输入的数据以字符串的形式返回。

[说明]: InputBox 函数显示一个能接受用户输入信息的对话框,对话框中有一个编辑框用来接收用户的输入。“对话框标题”显示在窗口的标题栏中,“信息内容”就是出现在对话框中的提示文本,“默认值”将显示在编辑框中作为用户输入的默认值。用户的输入以字符串的形式返回。

为了在窗体上显示文本,可使用窗体的 Canvas.TextOut 方法,该方法用来在窗体的指定位置显示成绩。

【程序代码】

```
implementation
var
    df:Array[1..2,1..4] of Integer;           //定义一个单元级数组变量
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    m,n:Integer;
begin
    Canvas.FillRect(Form1.ClientRect);        //清除窗体上显示的成绩
    for m:=1 to 2 do
        for n:=1 to 3 do
            begin
                df[m,n]:=StrToInt(InputBox('请输入成绩','输入第'+IntToStr(m)+'
                '个学生'+第'+IntToStr(n)+'门课成绩','0'));        //输入学生成绩
                Canvas.TextOut(30+40*n,30+30*m,IntToStr(df[m,n]));    //在指定位置显示成绩
            end;
            Canvas.TextOut(55,30,'语文');        //指定位置显示各门功课和学生甲乙
            Canvas.TextOut(95,30,'数学');
            Canvas.TextOut(135,30,'英语');
            Canvas.TextOut(175,30,'总成绩');
            Canvas.TextOut(25,60,'甲');
            Canvas.TextOut(25,90,'乙');
        end;
    procedure TForm1.Button2Click(Sender: TObject);
    var
        m,n:Integer;
    begin
        df[1,4]:=0;        //在统计总成绩前使其等于零
        df[2,4]:=0;
        for m:=1 to 2 do
            for n:=1 to 3 do
                df[m,4]:=df[m,4]+df[m,n];
            Canvas.TextOut(190,60,IntToStr(df[1,4]));
            Canvas.TextOut(190,90,IntToStr(df[2,4]));
        end;
```

4.1.4 动态数组的定义与使用

1. 动态数组的定义

静态数组在使用之前其类型及元素个数均已确定,而动态数组没有指定数组元素的个数,因此在程序运行时可为动态数组动态地开辟存储空间。一维动态数组的定义格式及功能如下。

[格式]:

Type

数组类型名=Array of 基类型

[功能]: 定义动态数组类型, 类型名由“数组类型名”指定。

[说明]: 与静态数组类型相比, 动态数组没有“下标类型”的定义。

定义了数组类型后, 就可以定义数组变量。与静态数组一样, 也可以将定义类型和变量合二为一。

多维动态数组的定义格式及功能如下。

[格式]:

Type

数组类型名=Array of Array of ... Array of (基类型)

[功能]: 定义多维动态数组类型, 类型名由“数组类型名”指定。

[说明]: 定义动态数组类型时, 有几个保留字“array”就是几维动态数组类型。

由于动态数组没有明确数组的大小, 在程序设计中可以调用 **SetLength** 标准过程来确定数组大小。例如, 下列程序段:

```
var
  arr:Array of Real;
begin
  SetLength(arr,10);
  ...
end;
```

定义了一个一维动态数组变量 **arr**, 数组元素的基类型为实型, 并且确定了该动态数组的大小为 10 个元素。

又如, 下列程序段:

```
var
  DyArr:Array of Array of Integer;
...
SetLength(DyArr,2,5);
...
```

定义了一个二维动态数组, 并确定其为 2 行 5 列。

注意: 关于动态数组, 应注意以下几点。

(1) 动态数组的下标是从零开始。

(2) 动态数组各行的长度可以不相等。

(3) 无论是静态数组还是动态数组, 都可以调用标准函数 **Low**、**High** 和 **Length** 来返回数组的最小下标值、最大下标值和数组的长度。

2. 动态数组的使用

下面通过实例来讲解动态数组的使用方法。

【例 4-3】 设计一个程序, 程序的设计界面如图 4-6 所示, 程序运行界面分别如图 4-7 和图 4-8 所示。程序运行时单击**【一维数组】**按钮, 将随机生成一个一维数组(长度为 1~9 中的某个数), 并显示; 单击**【二维数组】**按钮, 将会随机生成一个具有两行的二维数组(每

行长度为 1~9 中的某个数), 两个数组的元素值都是 100 以内的非负整数。



图 4-6 程序设计界面

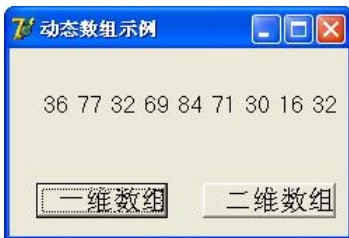


图 4-7 程序运行界面 (一)

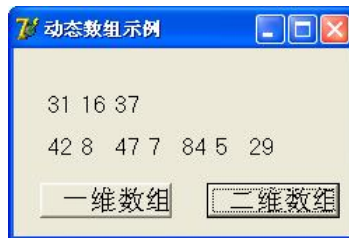


图 4-8 程序运行界面 (二)

【实现分析】

首先应该明白如何定义动态数组变量, 定义之后如何随机确定一维数组的元素个数。为生成数组的长度, 可利用随机函数 `Random` 函数, 例如, “`SetLength(Arr, Trunc(1+9* Random));`” 语句的功能是确定 `Arr` 数组的长度, 长度为 1~9 之间的随机数。对于每行长度随机的二维数组 (两行), 可以借助语句 `SetLength(Arr, 2)` 确定数组的行数为两行, 然后再确定每一行的长度, 确定每行的长度也通过产生一个 1~9 之间的随机数来实现。

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R_Arr:Array of Integer;           //定义一维动态数组, 注意下标是从零开始
  k:Integer;
begin
  Canvas.FillRect(Form1.ClientRect); //清除先前生成的数组元素
  SetLength(R_Arr,Trunc(1+9*Random)); //一维数组行的长度为 1~9
  for k:=Low(R_Arr) to High(R_Arr) do
  begin
    R_Arr[k]:=Trunc(100*Random);    //数组元素为 0~99
    Canvas.TextOut(24*(k+1),30,IntToStr(R_Arr[k])); //指定位置显示数组
  end;
end;
procedure TForm1.Button2Click(Sender: TObject);
var
  R_Arr:Array of Array of Integer; //定义二维动态数组, 注意下标从零开始
  m,n:Integer;
begin
  Canvas.FillRect(Form1.ClientRect); //清除先前生成的数组
  SetLength(R_Arr,2);                //动态数组行数为 2, 每行的长度还没有固定
  SetLength(R_Arr[0],Trunc(1+9*Random)); //第 0 行长度为 1~9
  SetLength(R_Arr[1],Trunc(1+9*Random)); //第 1 行长度为 1~9
  for m:=0 to 1 do
  for n:=Low(R_Arr[m]) to High(R_Arr[m]) do
  begin
    R_Arr[m,n]:=Trunc(100*Random); //生成 0~99 之间的随机数作为元素的值
```

```

        Canvas.TextOut(24*(n+1),30+30*m,IntToStr(R_Arr[m,n]));
    end;
end;

```

4.2 典型实例

4.2.1 典型实例一

【实例题目】：冒泡法排序

编写一个冒泡法排序程序，程序的设计界面如图 4-9 所示。程序运行时，单击【输入】按钮，从键盘上输入任意 10 个整数并显示在第一个编辑框中，然后单击【冒泡排序】按钮，10 个整数将按照从小到大的顺序重新排列，如图 4-10 所示。



图 4-9 程序设计界面

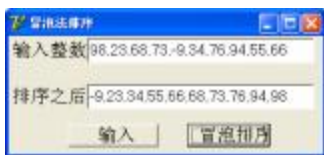


图 4-10 程序运行界面

【实现方法】

冒泡法是一种很重要的排序方法，假设要对 N 个数进行冒泡法排序，需要进行 $N-1$ 轮比较，各轮比较方法如下。

第 1 轮：将第 1 个数与第 2 个数比较，若第 1 个数大于第 2 个数则交换它们的位置，否则不交换；再将第 2 个数和第 3 个数比较，若第 2 个数大于第 3 个数则交换它们的位置，否则不交换；依此类推，直到第 $(N-1)$ 个数和第 N 个数比较，若前者大于后者，则交换它们的位置。经过第 1 轮 $N-1$ 次比较后，将最大的数放在了第 N 个数的位置，即第 N 个数排好。

第 2 轮：按照第 1 轮的方法，将第 1 个数与第 2 个数比较，若前者大于后者则交换它们的位置，再将第 2 个数与第 3 个数比较……直到第 $(N-2)$ 个数与第 $(N-1)$ 个数比较，若前者大于后者，则交换它们的位置，这样第 $(N-1)$ 个数排好。

……

第 $(N-1)$ 轮：将第 1 个数与第 2 个数比较，这样第 2 个数排好，剩下的第 1 个数自然而然也排好了。

因此，对 N 个数用冒泡法排序（从大到小），可以得到如下结论。

- ┃ 需要比较 $N-1$ 轮。
- ┃ 在第 K 轮比较中，需要比较 $N-K$ 次。
- ┃ 每轮总是从第 1 个元素和第 2 个元素开始比较，接着是第 2 个元素和第 3 个元素开始比较，依此类推。
- ┃ 比较的基本规则是：若前面的数大于后面的数则交换，否则不交换。


```

        n:=num[j];
        num[j]:=num[j+1];
        num[j+1]:=n;
    end;
end;
for i:=1 to 10 do
    Edit2.Text:=Edit2.Text+FloatToStr(num[i])+',';
    Edit2.Text:=Copy(Edit2.Text,1,Length(Edit2.Text)-1);    //去掉最后一个逗号
end;

```

4.2.2 典型实例二

【实例题目】：二维数组的“鞍点”

所谓二维数组的“鞍点”是指在本行中最大但在本列中最小的数组元素，有的数组有“鞍点”，有的数组没有。编程寻找一个从键盘上输入的3行4列数组的“鞍点”。程序设计界面如图4-11所示，程序运行界面如图4-12和图4-13所示。



图 4-11 程序设计界面



图 4-12 程序运行界面（一）



图 4-13 程序运行界面（二）

【实现方法】

寻找鞍点的方法是：根据鞍点的定义，首先确定第一行中的最大值，然后找到最大值的列坐标，再将该列的所有元素与刚才找到的最大值比较，如果该最大值是所在列中的最小值，则找到鞍点，否则该行没有鞍点。以此方法判断并寻找第二行第三行等其他行的鞍点。本例找到一个鞍点即终止寻找。

【程序代码】

```

implementation
var
    arr:Array[1..3,1..4] of Integer;    //定义单元级二维数组变量 arr
    {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    m,n:Integer;
begin
    Canvas.FillRect(Form1.ClientRect);    //清除先前产生的随机数组

```

```

for m:=1 to 3 do
  for n:=1 to 4 do
    arr[m,n]:=StrToInt(InputBox('输入数组元素','输入 arr['+IntToStr(m)+
      ','+IntToStr(n)+']:','0'));           //输入数组元素 arr[m,n]
  for m:=1 to 3 do                          //循环嵌套
    for n:=1 to 4 do
      Canvas.TextOut(20+35*n,40+25*m,IntToStr(arr[m,n]));      //显示 arr[m,n]
    end;

procedure TForm1.Button2Click(Sender: TObject);
var
  m,n,k,j,f:Integer;
begin
  for m:=1 to 3 do
    begin
      f:=1;                                //标记该行是否有鞍点，值为 1 表示有鞍点
      k:=1;
      for n:=2 to 4 do
        if arr[m,k]<arr[m,n] then
          k:=n;                            //寻找本行中最大值的下标
      for j:=1 to 3 do
        if arr[j,k]<arr[m,k] then
          begin
            f:=0;                          //本行中最大值元素不是该列最小值，故该行没有鞍点
            break;                          //终止循环
          end;
        if f=1 then
          break;                            //找到一个鞍点即终止
      end;
      if f=1 then
        Canvas.TextOut(30,30,'第一个鞍点坐标为 ('+IntToStr(m)+','+
          IntToStr(k)+') ')                //指定位置显示鞍点坐标
      else
        Canvas.TextOut(30,30,'没有鞍点');  //指定位置显示没有鞍点
    end;
end;

```

4.3 上机练习

4.3.1 上机练习一

【练习题目】：选择法排序

编写一个选择法排序程序，程序的设计界面如图 4-14 所示。程序运行时单击**【输入整数】**按钮，将允许用户输入 5 个整数，并以逗号隔开的形式显示在第一个编辑框中。单击**【选择**

法排序】按钮，则排好序的5个整数将显示在第二个编辑框里（从小到大排列），程序运行界面如图4-15所示。



图 4-14 程序设计界面



图 4-15 程序运行界面

【要点提示】

选择法排序也是重要的排序方法。假设要对 N 个数用选择法排序（从小到大排列），那么一共要进行 $N-1$ 轮，每轮的比较方法如下。

第1轮：从第1~ N 个数中选出最小的数和第1个数交换，排列好第1个数；

第2轮：从第2~ N 个数中选出最小的数和第2个数交换，排列好第2个数；

.....

第 i 轮：从第 i ~ N 个数中选出最小的数和第 i 个数交换，排列好第 i 个数；

.....

第 $N-1$ 轮：从第 $(N-1)$ ~ N 个数中选出最小的数和第 $(N-1)$ 个数交换，排列好第 $(N-1)$ 个数。当然最后1个数就是最大的，不用再排列，整个排序工作结束。

【参考代码】

```
implementation
const
    N=5;                                //定义一个常量 N，用来表示数组元素个数
var
    arr:Array[1..N] of Integer;         //定义单元级数组变量
    {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    i,j,p,t:Integer;
begin
    for i:=1 to N-1 do                  //一共进行 N-1 轮比较
    begin
        p:=i;                          //最小值下标为 i
        for j:=i+1 to N do              {第 i 轮比较时从第 i 个数到第 N 个数中
            寻找出最小数，用最小数与第 i 个数交换位置}
            if arr[p]>arr[j] then
                p:=j;                   //较小值下标赋值给 p
        if p<>i then                    //一轮比较完，将最小的数与第 i 个数交换排列好第 i 个数
        begin
            t:=arr[p];
```



```

        arr[p]:=arr[i];
        arr[i]:=t;
    end;
end;
for i:=1 to N do
    Edit2.Text:=Edit2.Text+IntToStr(arr[i])+',';           //每两个数之间以逗号分隔
    Edit2.Text:=Copy(Edit2.Text,1,Length(Edit2.Text)-1);   //去掉最后一个逗号
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    m:Integer;
begin
    Edit1.Text:='';                                         //清除编辑框数据
    Edit2.Text:='';
    for m:=1 to N do
        arr[m]:=StrToInt(TextBox('输入 N 个整数','输入 arr('+
            IntToStr(m)+')','0'));                         //输入数组元素 arr[m]
    for m:=1 to 5 do
        Edit1.Text:=Edit1.Text+IntToStr(arr[m])+',';
    Edit1.Text:=Copy(Edit1.Text,1,Length(Edit1.Text)-1);
end;

```

4.3.2 上机练习二

【练习题目】：求学生的平均成绩

某班有 M 名同学，本学期开了 N 门课，期末考试后，要统计每个学生的平均分。请编写一个程序实现该功能，程序的设计界面如图 4-16 所示，程序运行时单击**【形成学号与成绩】**按钮，将生成有序的学号并随机产生成绩，然后单击**【计算平均分】**按钮，将计算出每个学生的平均分，如图 4-17 所示。



图 4-16 程序设计界面



图 4-17 程序运行界面

【要点提示】

为记录学生的学号和成绩，可定义一个具有 M 行 N+1 列的二维数组，其中第 1 列用来存放学生的学号，其他列用来存放学生的成绩。由于每个学生要求出一个平均分，M 个同学就要求出 M 个平均分，因此可定义一个具有 M 个元素的一维数组来存放 M 个学生的平均分。本题求解的二维数组的模型可用图 4-18 表示。图中的 Cj 数组用来存放学号和成绩，Aver 数

组用来存放每个学生的平均分。

通过该图可以看出，第*i*个人的平均成绩可用下式求得：

$$\text{Aver}(i) = (C_j(i,2) + C_j(i,3) + \dots + C_j(i,N+1)) / N$$

该式子是一个求和的式子，可以用一个循环来实现。

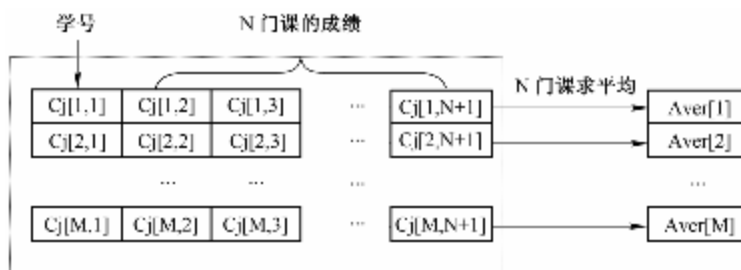


图 4-18 求 M 个学生 N 门课的平均成绩示意图

【参考代码】

```
implementation
const
    M=3;                                //常量，表示人数
    N=5;                                //常量表示课程门数
var
    Cj:array[1..M,1..N+1] of integer;   //记录学号和成绩的数组
    Aver:Array[1..M] of real;            //存放平均成绩的数组
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    i,j,x,y:integer;
begin
    Randomize;                          //随机数初始化
    For i:=1 to M do
        begin
            Cj[i,1]:=i;                  //生成学号
            For j:=2 to N+1 do
                Cj[i,j]:=Trunc(90*Random+10); //生成每个学生的每门课成绩
            end;
            Form1.Canvas.textout(1,1,'学号   成绩 1   成绩 2   成绩 3   成绩 4   成绩 5   平均分');
            For i:= 1 to M do
                //此二重循环用来输出学生的学号与各门课成绩
                For j:=1 to N+1 do
                    begin
                        x:=(j-1)*40;y:=(i)*20;
                        Form1.Canvas.textout(x,y,inttostr(Cj[i,j]));
                    end;
                end;
            end;
        end;
    procedure TForm1.Button2Click(Sender: TObject);
```

```

var
  i,j:integer;
begin
  For i := 1 To M do           //此循环用来求出每个学生的各门课的平均分
  begin
    Aver[i]:=0;
    For j:= 2 To N + 1  do
      Aver[i] := Aver[i] + Cj[i, j];
    Aver[i]:=Aver[i]/N;
    Form1.Canvas.textout((N+1)*40,i*20,Floattostr(Aver[i]));
  end;
end;

```

课后考场

一、选择题（20分，每题5分）

- 定义一个一维静态数组，该数组有5个元素，每个元素的类型为实型，下面正确的定义是_____。

A. Var arr:Array[1..5]of Integer;	B. Var arr:Array of Real;
C. Var arr:Array[1..5]of Real;	D. Var arr:Array of Real; SetLength(arr,5);
- 动态数组和静态数组的区别是_____。

A. 是否是一维数组	B. 元素类型是否是整型
C. 定义时是否明确了数组元素的个数	D. 元素类型是否一致
- 假设 arr 为一维静态数组，那么 Low(arr)表示_____。

A. arr 的前趋值	B. arr 的最小下标值
C. arr 的后继值	D. arr 的最大下标值
- 下面的 Button1Click 事件发生后，Edit1 中显示的文本是_____。

```

procedure TForm1.Button1Click(Sender: TObject);
var
  i:integer;
  a:array [0..9] of integer;
begin
  for i:=9 downto 0 do
    a[i]:=10-i;
  edit1.Text :=inttostr(a[2])+inttostr(a[5])+inttostr(a[8]);
end;

```

- | | | | |
|--------|--------|--------|--------|
| A. 258 | B. 741 | C. 852 | D. 369 |
|--------|--------|--------|--------|

二、填空题（40分，每空5分）

- 数组是一些具有_____类型的元素按一定顺序组成的序列。
- 下面的 Button1Click 事件发生后，Edit1 中显示的文本是_____。

```

procedure TForm1.Button1Click(Sender: TObject);
var
  i,j,s:integer;
  a:array [0..2,0..2]of integer;
Begin
  s:=1;
  for i:=0 to 2 do
    for j:=0 to 2 do
      a[i,j]:=i*3+j;
  for i:=0 to 2 do
    s:=s*a[i,2-i];
  edit1.Text :=inttostr(s);
end;

```

3. 动态数组的下标是从_____开始。
4. 假设 num 为一维静态数组, 有 5 个元素, 第一个元素是 num[1], 那么 num[5]表示_____。
5. 假设 arr 为二维动态数组, 那么语句 SetLength(arr,3)表示数组 arr 行数等于_____, 要使第一行有五列, 使用的语句为_____。
6. 语句 “Var arr:Array of Array of Integer;” 表示定义了一个_____维动态数组。
7. 已知定义了一个一维静态数组 a, 要获取它的下标上界, 函数调用为_____。

三、程序设计题 (40 分, 每题 20 分)

1. 编写一个求矩阵转置的程序。矩阵的转置就是把第一行的元素变成第一列的元素, 第二行的元素变成第二列的元素, 依此类推。程序的设计界面如图 4-19 所示。程序运行时单击【随机矩阵】按钮将产生一个 3 行 5 列的随机矩阵, 矩阵元素为 100 以内的非负整数; 单击【转置矩阵】按钮产生该矩阵的转置, 如图 4-20 所示。



图 4-19 程序设计界面



图 4-20 程序运行界面

2. 设数组 a 中的元素均为正整数, 请编程求出该数组中奇数的个数和偶数的个数。程序设计界面如图 4-21 所示, 程序运行界面如图 4-22 所示。



图 4-21 程序设计界面



图 4-22 程序运行界面

第 5 章 过程与函数

本章要点

- ▮ 过程与函数的概念
- ▮ 过程与函数的定义方法
- ▮ 过程与函数中三种参数传递的概念及特点
- ▮ 过程与函数的调用方法

5.1 理论知识

过程与函数就是完成某一特定功能的一段程序，即子程序，过程没有返回值，函数有返回值。使用过程与函数的第一个原因是结构化程序设计的需要，同时过程和函数也是结构化程序设计的重要手段。结构化程序设计思想最重要的一点就是把一个复杂的问题分成很多小而独立的问题，即把一个大程序分为若干小程序——模块，每个模块完成一部分功能。如图 5-1 所示。对于每个模块，需要详细定义模块的功能及其接口，一个程序员编制其中的一个或多个模块，并把模块编写成函数或过程。模块编写好后，可以把它们组装成应用程序，如把“二级子模块 11”和“二级子模块 12”组合在一起就实现了“一级子模块 1”的功能，把“二级子模块 21”和“二级子模块 22”组合在一起就实现了“一级子模块 2”的功能，把“一级子模块 1”和“一级子模块 2”组合在一起就实现了软件项目的功能。组合子模块可以通过调用相应的过程或函数来实现。

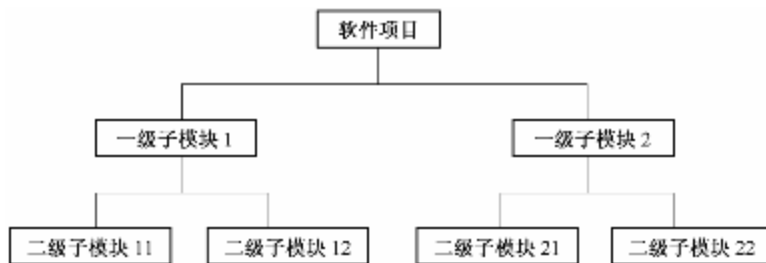


图 5-1 结构化程序设计模式

使用过程或函数的第二个原因是为了解决代码的重复。可以把经常用到的完成某种功能的程序段编写成过程或函数，这样做有很多好处，例如：每当需要完成这一功能时只要调用这个过程或函数即可，而不需重复编写代码；如果需要修改这一段代码，只要在该过程或函数里修改即可，而调用该过程或函数的程序不必修改。

5.1.1 过程与函数的概念

1. 过程

Delphi 语言的过程有两种：标准过程与自定义过程。标准过程是系统内部定义好的过程，用户不必编写任何代码即可拿来使用，例如插入子字符串过程 `Insert`、字符串复制过程 `Copy` 等。自定义过程由用户自己编写代码定义，它又分为事件过程与通过程两种。事件过程依附于特定的对象，如按钮的单击事件过程，事件过程的建立方法在前面已经介绍过，此处不再赘述。通过程用于完成某个特定的功能，必须由别的过程（如事件过程）调用才能执行。为便于说明，把被其他过程调用的过程（或函数）称为被调过程（或函数），把调用其他过程（或函数）的过程（或函数）称主调过程（或函数）。

2. 函数

函数也是完成某一功能的程序段，与过程最重要的区别在于，函数有返回值而过程没有。Delphi 的函数也有两类：标准函数和自定义函数。标准函数是系统内部定义好的可以直接调用的函数，例如 `Sin(x)`、`Sqr(x)` 等。自定义函数是需要用户自己编写代码实现的函数。

5.1.2 过程的定义与调用

1. 过程的定义

事件过程很简单，例如在窗体中添加一个按钮 `Button1`，双击该按钮，Delphi 将自动产生一个事件过程 `Button1Click`。事件过程名也有规律，那就是组件名加事件类型名（省略了 `On`）或 `Form` 加事件类型名称（注意不是窗体名，如 `Form1` 窗体的创建事件过程名为 `FormCreate`，而不是 `Form1Create`）。事件过程在发生相应事件时被调用执行。

通过程由用户自己创建并通过程序调用，创建通过程也称过程声明或过程定义。通过程的定义格式与功能如下。

[格式]:

<code>Procedure</code>	过程名 ([形参表])	// <code>Procedure</code> 表示这是一个过程，有形参也
		//可以没有
	局部声明	//声明常量、变量或另一个过程或函数等
<code>begin</code>		
	语句;	//简单语句或复合语句都可以
<code>end;</code>		

[功能]: 定义通过程，过程名由“过程名”指定。

[说明]:

(1) 通过程分为过程首部和过程体，以保留字 `Procedure` 开头的这一行是过程首部，其余的是过程体。

(2) 过程首部的说明。“过程名”是任何合法的标识符。“形参表”包括若干个形式参数，也可以没有形参，形式参数又简称形参，是定义过程或函数时使用的参数。实际参数又称实参，是调用过程或函数时实际使用的参数。形参表指定了调用过程传递给被调过程（用户定义的通过程）的参数（实际参数）的个数与类型。如果有形参，必须指定形参的类型。在

调用过程时，实参的个数、位置、类型与含义必须与形参一一对应。

(3) 过程体说明。过程体是一个程序段。其中在“局部声明”部分定义的类型、常量和变量的作用范围是局部的，即只能在该过程内部使用。**begin** 与 **end** 之间的部分为执行部分，可以是各种可执行语句。

(4) 形参说明的格式与功能如下。

[格式 1]:

形参名:类型

[格式 2]:

var 形参名:类型

[格式 3]

const 形参名:类型

[功能]: 为过程说明一个形式参数，形式参数的名称由“形参名”指定，形式参数的类型由“类型”指定。

[说明]: 保留字 **var** 表示该形参为变量参数，简称变参，相应的变量实参将按地址方式传递到形参，因此对形参的改变将影响实参值。保留字 **const** 表示形参在过程中是常量参数，在函数中不能改变它的值。如果无 **var** 和 **const**，表示在过程中可以改变该形参的值，但过程调用完成后，并没有改变对应的实参值，这种形参又叫值参。

2. 过程的调用

创建的通用过程是为了调用的，通用过程只有通过调用才能被执行。若想要调用某个过程，该过程必须在调用语句之前定义。过程调用语句的格式与功能如下。

[格式 1]:

过程名(实参表);

[功能]: 对已经定义的通用过程进行调用，该过程有形参。

[格式 2]:

过程名; 或者: 过程名();

[功能]: 调用已经定义的通用过程，该过程无形参。

[说明]: 通用过程没有形参时采用格式 2 进行调用，否则采用格式 1。有形参时，调用时实参的个数、位置和类型必须与形参一一对应，实参之间以逗号分隔。

【例 5-1】 定义一个过程，用来在一个 **Memo** 组件中显示若干行欢迎词，欢迎词的显示次数是随机的，但至少显示 3 次，至多显示 7 次。程序的设计界面如图 5-2 所示，程序运行时，单击**【显示欢迎词】**按钮，将在 **Memo** 组件中显示出若干行欢迎词，如图 5-3 所示。

【实现分析】

根据题意分析可知，显示欢迎词其实就是把欢迎词连接到 **Memo** 组件的 **Text** 属性中，只是若干个操作而已，并没有返回结果，因此适合用过程来实现。由于欢迎词的显示次数是不确定的，因此过程应有一个形参用来接收传过来的显示次数。显示次数可通过随机数来实现。

【界面设计】

窗体组件属性设置及其作用如表 5-1 所示。



图 5-2 程序设计界面



图 5-3 程序运行界面

表 5-1 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Memo1	Font.Size Font.Color	12 clNavy	显示欢迎词
Button1	Caption	'显示欢迎词'	单击它将显示不定次数的欢迎词

【程序代码】

```
procedure Welcome(n:integer);           //参数 n 用来接收欢迎词的显示次数
var
  i:integer;
begin
  For i:=1 to n do                      //显示 n 次欢迎词
  begin
    Form1.Memo1.text:=Form1.Memo1.text+'欢迎你进入 Delphi 世界! '+char(13)+char(10);
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  n:integer;
begin
  Randomize;                           //随机数初始化
  n:=trunc(3+5*Random);                 //产生 3~7 之间的随机数
  Welcome(n);                           //调用过程显示欢迎词
end;
```

5.1.3 函数的定义与调用

1. 函数的定义

函数也是完成一定功能的程序段，与过程的主要区别就是函数有返回值，其定义方式稍有不同，函数的定义格式及功能如下。

[格式]:

Function 函数名（形参表）:返回值类型;
局部声明


```
begin  
    语句;  
end;
```

[功能]: 定义一个函数, 函数名由“函数名”指定。

[说明]:

(1) 格式中第一行为函数首部, 形参表的含义与过程完全一致。“返回值类型”规定了函数返回值的数据类型, 返回值可以通过给函数名赋值或给预定义的隐含变量 **Result** 赋值来实现。

(2) 除第一行外的其他部分为函数体, 含义同过程体。有一点需要注意, 在语句中至少要给函数名或 **Result** 赋值一次, 以便让函数执行完毕时把函数值带回给主调过程或主调函数。

2. 函数的调用

函数也是通过调用来运行的, 调用函数的一般格式如下。

[格式 1]:

变量名:=函数名(实参列表);

[功能]: 调用“函数名”指定的函数, 并把函数的返回值赋值给变量。

[格式 2]:

表达式 运算符 函数名(实参列表);

[功能]: 调用“函数名”指定的函数, 得到一个值, 该值与“表达式”的值进行“运算符”指定的运算, 得到一个结果。

注意: 函数有一个返回值, 函数调用就相当于一个同类型的值, 可以和同类型的数据一样作为表达式或表达式的一部分参加运算。

【例 5-2】 定义一个函数, 用来求任意非负整数各位数字之和。程序设计界面如图 5-4 所示, 程序运行时, 在第一个编辑框中输入一个整数, 然后单击【计算】按钮, 将该整数的各位数字和求出来并显示在第二个编辑框中, 如图 5-5 所示。要求把求整数的各位数字和编写成一个函数。



图 5-4 程序设计界面



图 5-5 程序运行界面

【实现分析】

要通过函数求整数的各位数字和, 函数应有一个参数用来接收传给它的整数, 函数应把最后的结果, 即整数的各位数字和, 作为函数值返回, 在函数中实现求整数的各位数字和的功能。假设要求 m 的各位数字和, 应反复执行下面三条语句: “ $r:=m \bmod 10;$ ”, “ $m:=m \div 10;$ ”和 “ $sum:=sum+r;$ ”, 直到 m 的值为 0 时为止, 此时的 sum 就是 m 的各位数字和。需要注意的是应把求得和赋值给函数名或 **Result** 变量才能作为函数的返回值。

【界面设计】

窗体组件属性设置及其作用如表 5-2 所示。

表 5-2 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'输入一非负整数: '	
Edit1	Text	"	输入一个整数
Label2	Caption	'各位数字和为: '	
Edit2	Text	"	显示求得的整数的各位数字和
Button1	Text	'计算'	单击它将计算出整数的各位数字和并显示在 Edit2 中

【程序代码】

```
Function weisum(m:Integer):Integer;           //注意自定义函数一般在事件过程之前定义
var
    sum,r:integer;
begin
    sum:=0;
    repeat
        r:=m mod 10;           //m 除 10 的余数(即 m 的个位数)赋值给变量 r
        m:=m div 10;           //m 除 10 的商再赋值 m
        sum:=sum+r;             //把取出的位加到和中去
    until(m=0);                 //直到 m 等于 0 时为止
    weisum:=sum;                 //把各位数字和赋值给函数名, 作为函数的返回值
end;
procedure TForm1.Button1Click(Sender: TObject);
var
    k:integer;                   //存放要求各位数字和的数
    s:integer;                   //存放求得的各位数字之和
begin
    k:=StrToInt(Edit1.Text);     //字符串转换成整数
    s:=weisum(k);
    Edit2.Text:=IntToStr(s);     //将各位数字之和显示在编辑框 2 中
end;
```

5.1.4 参数的传递

过程与函数的形参有三种：变量参数、常量参数和值参。在过程调用时，无论是哪种参数，都要求实参与形参在个数、位置、类型与含义上一一对应，即第一个实参传递给第一个形参，第二个实参传递给第二个形参，依此类推。

1. 常量参数和值参的传递

在过程或函数的首部中，定义形参时，如果形参名前有保留字 **const**，说明该参数是常量参数。在函数或过程中不允许给常量参数赋值。在定义形参时，如果形参前没有任何保留字，

则说明该参数为值参，在函数或过程中可以给值参赋值，但在调用该函数或过程时不会改变与之对应的实参的值。值参的这种传递方式称为值传递，值传递是一种单向传递，即可把实参值传递给形参，但对形参的改变不会传给实参。

【例 5-3】 编写一个求两个数的乘积的应用程序，程序设计界面如图 5-6 所示。程序运行时，在“操作数 1”编辑框输入任意整数，在“操作数 2”编辑框也输入任意整数，单击【运算】按钮将把两个操作数的乘积运算出来，并显示在最下面一个编辑框里，如图 5-7 所示。要求把求两个数的乘运算编写成一个通用过程。



图 5-6 程序设计界面

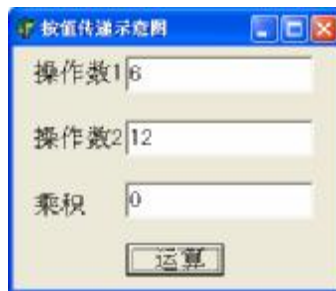


图 5-7 程序运行界面

【实现分析】

把乘积运算作为一个通用过程，具体进行乘积运算时只要调用该过程即可。通用过程应有 3 个参数：操作数 1、操作数 2 和乘积结果。假定这 3 个参数用 k1、k2 和 k 表示，那么自定义通用过程的首部就可以写成：“Procedure cj(k1,k2,k:Integer);”。在【运算】按钮的单击事件代码中调用该过程以便计算出乘积。

【界面设计】

窗体组件属性设置及其作用如表 5-3 所示。

表 5-3 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'操作数 1'	提示输入操作数
Label2	Caption	'操作数 2'	提示输入操作数
Label3	Caption	'乘积'	提示作乘积运算
Edit1	Text	"	用来输入操作数
Edit2	Text	"	用来输入操作数
Edit3	Text ReadOnly	" True	用来显示乘积
Button1	Caption	'运算'	单击将作乘积运算
Form1	Caption	'按值传递示意图'	容纳其他组件

【程序代码】

```

procedure cj(k1,k2,k:Integer);           //自定义过程进行乘积运算
begin
    k:=k1*k2;                           //乘积赋值给 k

```

```

end;
procedure TForm1.Button1Click(Sender: TObject);
var
    m1,m2,m:Integer;
begin
    m:=0;                                //给变量赋初值
    m1:=StrToInt(Edit1.Text);  m2:=StrToInt(Edit2.Text);
    cj(m1,m2,m);                    //调用过程求乘积
    Edit3.Text:=IntToStr(m);        //将结果用编辑框显示出来
end;

```

从图 5-7 中可以看出,程序的运行并没有得到期望的结果!错误的原因在于值传递,即定义过程时,各形参前没有保留字,所以均为值参。在调用时把 m 的值传给了形参 k ,在过程中对 k 的值进行了修改但并没有修改实参 m 的值, m 的值依旧是 0!如果希望通过参数来得到两个数的乘积,就需要使用变量参数。

2. 变量参数(变参)的传递

在定义过程或函数时,形参表中 `var` 之后的参数即是变参。变参是按地址传递的,即在调用过程或函数时,实参的地址被传递给形参,故形参和实参实际上是同一个地址单元。因此在过程或函数中,被改变了的形参值在退出过程或函数时将保存在相应的实参中。可见,变参是一种双向的参数传递,实参值能够传递给形参,形参值也可以传给实参。

【例 5-4】 实现例 5-3 中的功能,得到正确的答案。
程序设计界面如图 5-6 所示,程序的运行界面如图 5-8 所示。

【实现分析】

可把存放乘积结果的形参定义为变参,这样在调用过程时就可以把乘积结果通过实参带回给主调过程或函数。

【界面设计】

窗体组件属性设置及其作用如表 5-3 所示。

【程序代码】

这里只列出稍有不同的通用过程代码,如下所示:

```

procedure cj(k1,k2:Integer;var k:Integer);    //形参 k 为变量参数
begin
    k:=k1*k2;                                //乘积赋值给 k
end;

```



图 5-8 程序运行界面

5.1.5 子程序的嵌套与递归

1. 子程序的嵌套

在 Delphi 中,过程与函数统称为子程序,子程序既能嵌套定义也能嵌套调用。嵌套定义的含义是,在一个子程序中的定义中又包含了另一个子程序的定义。嵌套调用是指在调用某个子程序时,该子程序又调用了其他的子程序。在嵌套定义时,被包含的子程序称为内层子

程序，包含其他子程序的子程序称为外层子程序。紧邻的两层称为邻层，否则称为隔层。关于子程序的嵌套定义和嵌套调用必须遵循如下规则。

- (1) 外层必须完全包含内层。
- (2) 外层可以调用内层中的邻层子程序，但不能隔层调用。
- (3) 内层可以调用外层中的邻层和隔层子程序。
- (4) 同一层的子程序允许后定义的子程序调用先定义的子程序，反之，需要用 Forward（超前引用）在子程序的首部后面对后定义的子程序予以说明才能调用。

【例 5-5】 组合数的计算公式为 $C_m^z = \frac{m!}{z!(m-z)!}$ 。请用子程序的嵌套来求组合数。程序设计界面如图 5-9 所示，程序运行界面如图 5-10 所示。

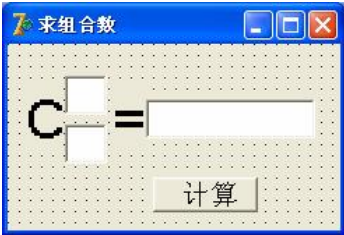


图 5-9 程序设计界面

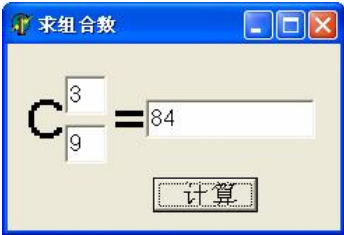


图 5-10 程序运行界面

【实现分析】

可以用一个子程序来求组合数。根据组合数 $C_m^z = \frac{m!}{z!(m-z)!}$ 可知，在求组合数时还必须用阶乘求出组合数式子中的分子和分母部分，而求阶乘也可以用一个子程序来实现。因此这是子程序的嵌套。

【界面设计】

窗体组件属性设置及其作用如表 5-4 所示。

表 5-4 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'C'	提示求组合数
Label2	Caption	'='	提示右边为组合数
Edit1	Text	"	用来输入数据
Edit2	Text	"	用来输入数据
Edit3	Text ReadOnly	" True	显示组合数
Button1	Caption	'计算'	单击求组合数
Form1	Caption	'求组合数'	容纳其他组件

【程序代码】

```
Function zh(k_fz,k_fm:Integer):Integer; //自定义外层函数求组合数
var //定义函数 zh 的变量
```

```

    fz, fm1, fm2: Integer;
Function  jc(k: Integer): Integer;           //内层函数求阶乘
var
    n1, n: Integer;                         //定义函数 jc 的变量
begin
    n:=1;
    for n1:=1 to k do
        n:=n*n1;
    jc:=n;
end;                                         //内层函数体结束
begin
    zh:=1;
    fz:=jc(k_fz);                           //调用内层子程序求组合数的分子部分
    fm1:=jc(k_fm);                           //调用内层子程序求组合数的分母部分
    fm2:=jc(k_fz-k_fm);
    zh:=fz div (fm1*fm2);                   //由于组合数肯定是个整数，因此这里用整除形式
end;
procedure TForm1.Button1Click(Sender: TObject);
var
    m1, m2, m: Integer;
begin
    m1:=StrToInt(Edit1.Text); m2:=StrToInt(Edit2.Text);
    m:=zh(m2, m1);                           //调用组合函数 zh 求组合数
    Edit3.Text:=IntToStr(m);
end;

```

2. 子程序的递归

子程序直接或间接调用自身称为子程序的递归调用。递归调用在处理数学上阶乘运算、级数运算等方面非常有效，有些看似很复杂的问题通过递归算法就变得很简单明了。不过，递归调用需要保留大量的中间状态，执行效率低。

【例 5-6】 有 6 个人坐在一块儿，问第 6 个人多大年纪？他说他比第 5 个人大 3 岁；再问第 5 个人多少岁，他说比第 4 个人大 3 岁；再问第 4 个人多少岁，他说比第 3 个人大 3 岁；问第 3 个人多少岁，他说比第 2 个人大 3 岁；问第 2 个人多少岁，他说比第 1 个人大 3 岁；最后问到第 1 个人多少岁，他说 12 岁。请问第 6 个人多少岁？用子程序递归调用来实现。程序设计界面如图 5-11 所示，程序运行界面如图 5-12 所示。

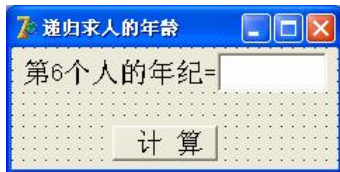


图 5-11 程序设计界面

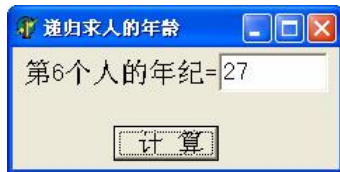


图 5-12 程序运行界面

【实现分析】

欲知道第 6 个人多少岁，必须知道第 5 个人多少岁，同样，欲知道第 5 个人多少岁，必须知道第 4 个人多少岁，依此类推。显然，这是个递归问题。假设 $A(1)$ 表示第 1 个人的年纪， $A(2)$ 表示第 2 个人的年纪， $A(n)$ 表示第 n 个人的年纪，可以用下面的式子表示每个人的年纪：

$$\begin{cases} A(6) = A(5) + 3 \\ A(5) = A(4) + 3 \\ A(4) = A(3) + 3 \\ A(3) = A(2) + 3 \\ A(2) = A(1) + 3 \\ A(1) = 12 \end{cases}$$

对上面的式子进行简化，如下所示：

$$\begin{cases} A(n) = 12, & n = 1 \\ A(n) = A(n-1) + 3, & n \geq 2 \end{cases}$$

从程序设计的角度来说，递归过程必须解决两个问题：一是递归计算的公式，二是递归结束的条件。本例可以写成：

递归计算公式： $A(n)=A(n-1)+3 \quad n \geq 1$

递归结束条件： $A(1)=1$

凡是能够表示成上述式子的数学问题均可以用递归来实现，在递归函数中一般可采用双分支语句来实现：

```
if (递归结束条件) then
    函数名:=递归终止值
else
    函数名:=递归公式;
```

式子中的函数名也可以用 **Result** 来代替。

【界面设计】

窗体组件属性设置及其作用如表 5-5 所示。

表 5-5 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption AutoSize	'第 6 个人的年纪=' False	提示这是第 6 个人的年纪
Edit1	Text ReadOnly	" True	显示第 6 个人的年纪
Button1	Caption	'计算'	单击计算第 6 个人的年纪

【程序代码】

```
Function A(n:Integer):Integer;           //自定义函数（形参为值参）
begin
    Case n of
        1:A:=12;
        else
```

```

        A:=A(n-1)+3;           //递归调用
    end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
    m,k:Integer;               //定义局部变量
begin
    m:=6;
    k:=A(m);                   //函数递归调用
    Edit1.Text:=IntToStr(k);   //整型数据转换成字符串
end;
```

5.2 典型实例

5.2.1 典型实例一

【实例题目】：验证哥德巴赫猜想

德国著名数学家哥德巴赫提出：任何一个大于等于 6 的偶数都可以表示为两个素数之和。请编写程序验证这一猜想。程序的设计界面如图 5-13 所示，程序运行时，在第一个编辑框里输入任意大于等于 6 的偶数，单击**【验证】**按钮，则在另一编辑框里显示两个符合要求的素数。程序运行界面如图 5-14 所示。要求判断某数是否为素数用一个函数来实现。

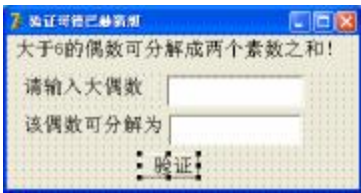


图 5-13 程序设计界面

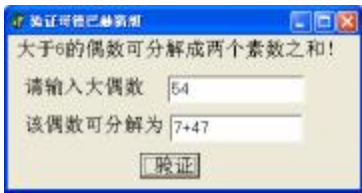


图 5-14 程序运行界面

【实现方法】

要用函数来判断某数是否为素数，应传给该函数一个要判断的数，函数的返回值应说明是还是不是，所以应该是布尔型。判断素数的方法已经介绍，此处不再赘述。为把某偶数 N 分解成两个素数，可通过循环依次判断 3~N/2 之间的每个奇数（假设为 K）是否为素数，若是素数，则判断 N-K 是否为素数，如果也是素数则分解成功，退出循环；否则看下一个奇数 K 和 N-K 是否同为素数；……；直到找到 K 和 N-K 同时为素数时为止。在循环中判断某数是否为素数可通过调用函数来实现。

【界面设计】

窗体组件属性设置及其作用如表 5-6 所示。

表 5-6 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'大于 6 的偶数可分解成两个素数之和 '	提示本程序的功能
Label2	Caption	'请输入大偶数'	提示输入符合条件的偶数
Label3	Caption	'该偶数可分解为'	提示该偶数可以表示为两个素数之和
Edit1	Text	"	用来输入符合条件的偶数
Edit2	Text	"	用来显示两个素数
Button1	Caption	'验证'	单击将显示符合条件的两个素数

【程序代码】

```

Function  prime(n:Integer):Bool;           //自定义函数，用来判断参数 n 是否为素数
var
    i:integer;
    flag:Bool;                             //标记是否为素数
begin
    flag:=true;                             //首先为 true
    for i:=2 to (n div 2) do
        if (n mod i=0) then                //不是素数
            begin
                flag:=false;               //标记为 false
                break;
            end;
    result:=flag;                           //把标记赋值给函数名
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    k,n:Integer;
    flag:bool;
begin
    n:=StrToInt(edit1.Text );              //输入的偶数
    k:=3;                                   //从 3 开始分解
    flag:=false;                            //标记有没有找到，一开始为 false，找到时为 true
    while(k<=(n div 2)) do
        begin
            if (prime(k) and prime(n-k)) then //如果找到
                begin
                    flag:=true;             //标记为 true
                    break;
                end;
            k:=k+2;                          //判断下一个奇数
        end;
    if (flag=true) then                      //如果分解成功
        Edit2.Text :=inttostr(k)+'+'+inttostr(n-k); //显示两个数
end;

```

5.2.2 典型实例二

【实例题目】：斐波那契（Fibonacci）数列

斐波那契（Fibonacci）数列是一个典型的可用递归求解的问题。该数列来源于兔子的繁殖问题，大意是：小兔子和大兔子没有繁殖能力，只有老兔子才有繁殖能力。假设第 1 个月有一对小兔子，第 2 个月长成大兔子，第 3 个月长成老兔子，并生出一对小兔子；第 4 个月，老兔子继续生出一对小兔子，同时原来的小兔子长成大兔子，这样在第 4 个月就有一对老兔子，一对大兔子和一对小兔子共 3 对兔子；依此类推，假设兔子永远不死，请编写程序求出任意一个月有多少对兔子。程序设计界面如图 5-15 所示，程序运行时输入月数，然后单击**【计算】**按钮，将显示出该月的兔子对数，如图 5-16 所示。



图 5-15 程序设计界面



图 5-16 程序运行界面

【实现方法】

本例直接求解很困难，但如果仔细观察就会发现：除了第 1 个月和第 2 个月只有一对兔子外，第 3 个月的兔子数是第 1 个月和第 2 个月兔子对数的和，第 4 个月是第 3 个月和第 2 个月兔子对数的和，以后的每个月都是前两个月兔子对数的总和。因此，可以用递归调用的方法来实现，递归的数学公式如下（假设第 n 个月有 $\text{Fib}(n)$ 对兔子）：

$$\text{Fib}(n) = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ \text{Fib}(n - 1) + \text{Fib}(n - 2), & n \geq 3 \end{cases}$$

【界面设计】

窗体组件属性设置及其作用如表 5-7 所示。

表 5-7 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'请输入月数'	提示输入月份
Label2	Caption	'该月兔子对数'	提示总兔子数
Edit1	Text	"	用来输入第几个月
Edit2	Text ReadOnly	" True	用来显示兔子数
Button1	Caption	'计算'	单击将计算出该个月兔子数

【程序代码】

```
Function Fibo(n:Integer):Integer;           //自定义函数求兔子数
```

```

Begin
  Case n of
    1:Fibo:=1;
    2:Fibo:=1;
    Else
      Fibo:=Fibo(n-1)+Fibo(n-2);          //递归函数
    end;
  end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  m:Integer;
begin
  m:=StrToInt(Edit1.Text);
  Edit2.Text:=IntToStr(Fibo(m));          //调用递归函数
end;

```

5.3 上机练习

5.3.1 上机练习一

【练习题目】：分离字母和数字字符

编写一个程序，用来分离出字符串中的字母字符和数字字符。程序的设计界面如图 5-17 所示。程序运行时，在第一个编辑框中输入任意一串字符，然后单击**【分离】**按钮，将把字母和数字分离开来并显示在对应的编辑框中，如图 5-18 所示。



图 5-17 程序设计界面



图 5-18 程序运行界面

【要点提示】

可定义两个函数分别用来判断某个字符串是否是数字或字母。要判断某字符串是否为数字字符，该函数应有一个形参，可定义为 `str`，用来接受要判断的字符串，当 `(str<='9') And (s >='0')` 为 `True` 时，是数字字符，函数返回值为 `True`，否则返回值为 `False`。函数的返回值应为 `Boolean` 型。同样的道理可编写判断某字符串是否为字母的函数。为分离出数字或字母，应一一取出输入字符串中的每个字符，再调用相应的判断函数测试是否是字母或数字，如果是，则把分离出来的字符连接到相应编辑框中。

【参考代码】

```
Function iszimu(s:String):Boolean;          //自定义判断字母的函数
```

```

begin
    if (s<='z')And(s>='a') Or (s<='Z')And(s>='A') then
        //注意上面逻辑表达式的运算顺序
        iszimu:=True
    else
        iszimu:=False;
    end;
Function isshuzi(s:String):Boolean;           //自定义判断数字字符的函数
begin
    if (s>='0')And(s<='9') then
        isshuzi:=True
    else
        isshuzi:=False;
    end;
procedure TForm1.Button1Click(Sender: TObject);
var
    t:String;
    m,n:Integer;
    b1,b2:Boolean;
begin
    Edit2.Text:=""; Edit3.Text:="";
    m:=Length(Edit1.Text);                     //将字符串的长度赋给 m
    for n:=1 to m do
        begin
            t:=Copy(Edit1.Text,n,1);           //每次截取一个字符
            b1:=isshuzi(t);                     //将数字函数的返回值赋值给 b1
            b2:=iszimu(t);                     //将字母函数的返回值赋值给 b2
            if b1 then
                Edit2.Text:=Edit2.Text+t;       //如果是数字就添加到数字编辑框
            if b2 then
                Edit3.Text:=Edit3.Text+t;       //如果是字母就添加到字母编辑框
        end;
    end;
end;

```

5.3.2 上机练习二

【练习题目】：校园歌手评分程序

为某次校园歌手大赛编写一个评分程序。设共有裁判6人，评分满分为10分，除去一个最高分，再除去一个最低分，剩余的评分的平均值即为选手得分。程序的设计界面如图5-19所示，程序运行时，分别在6个编辑框中输入6个评委的评分，然后单击【得分】按钮，将算出该选手的得分并在相应的编辑框中显示出来，如图5-20所示。单击【清除】按钮，将清除掉所有编辑框中的内容。要求：编写一个过程求选手得分，得分通过参数返回给调用过程。



图 5-19 程序设计界面



图 5-20 程序运行界面

【要点提示】

可先定义一个由 6 个元素组成的数组，用来存放评委的评分。程序运行时，把输入的评委评分存放到该数组中。通过分析可知，求选手得分的过程有两个参数，一个用来接收评委的评分，另一个用来返回选手的得分。接收成绩的数组，可以定义成一个动态数组，注意它的下标是从 0 开始的。由于要把选手的得分返回到调用过程，所以存放选手得分的参数的传递方式是“引用传递”，即为变参。在过程中通过一个循环把评分数组的各元素值加到一个和变量中并求出评分数组的最大值和最小值，然后用和变量的值减去最大值和最小值再除以 4，把求得的结果赋值给返回选手得分的参数即可。

【参考代码】

```
implementation
const
    N=6;
{$R *.dfm}
//根据参数 s 数组中存放的评分用来求选手的最后得分，并通过参数 aver 带回
procedure GetScore(S:array of real; var aver:real);
var
    i:integer;
    max_s,min_s:real;
    Sum:real;
begin
    Sum:=0;                                //分数和
    max_s:=s[0];min_s:=s[0];               //最高得分和最低得分为第一个评分
    For i:=0 to N-1 Do                    //该循环求最高得分和最低得分并把所有得分加起来
    begin
        if max_s<s[i] then
            max_s:=s[i];
        if min_s>s[i] then
            min_s:=s[i];
        sum:=sum+s[i];
    end;
    aver:=(sum-min_s-max_s)/(N-2);          //求平均分
end;
procedure TForm1.Button1Click(Sender: TObject); //得分按钮
var
    ssdf:real;
```

```

Scores:array[0..N-1] of real;           //存放得分的数组
begin
  Scores[0]:=strtoint(edit1.text);  Scores[1]:=strtoint(edit2.text);
  Scores[2]:=strtoint(edit3.text);  Scores[3]:=strtoint(edit4.text);
  Scores[4]:=strtoint(edit5.text);  Scores[5]:=strtoint(edit6.text);
  GetScore(Scores,ssdf);           //调用过程
  Edit7.Text:=floattostr(ssdf);
end;
procedure TForm1.Button2Click(Sender: TObject);    //清除按钮
begin
  edit1.Text:=""; edit2.Text:=""; edit3.Text:="";
  edit4.Text:=""; edit5.Text:=""; edit6.Text:=""; edit7.Text:="";
end;

```

课后考场

一、选择题（20分，每题5分）

- Delphi的标准过程是指_____。
 - 事件过程
 - 通用过程
 - 系统内部定义的过程
 - 用户编写代码定义的过程
- 在 Delphi 中关于子程序的说明正确的是_____。
 - 既不允许子程序嵌套定义，也不允许子程序嵌套调用
 - 不允许子程序嵌套定义，但允许子程序嵌套调用
 - 允许子程序嵌套定义，但不允许子程序嵌套调用
 - 既允许子程序嵌套定义，也允许子程序嵌套调用
- 下列过程首部定义正确的是_____。
 - procedure Fun1(k:real;s:real;n:integer);
 - procedure Fun1(k:real,s:real,n:integer);
 - procedure Fun1(k;s:real,n:integer);
 - procedure Fun1(Real:k;s,Integer:n);
- 已知过程首部如下：

```
procedure Fun1(var k:Integer;s:Integer;n:Integer);
```

设 A、C 是整型变量，则下列调用中，一定正确的是_____。

- Fun1(A,5,7)
- C=A+Fun1(A,5,7)
- Fun1(5,A,7)
- C=A+Fun1(5,A,7)

二、填空题（40分，每空5分）

- 过程与函数的本质是相同的，区别主要是_____。
- 在 Delphi 中，定义形参时，如果在形参前加保留字_____表示该参数是变量参数，如果在形参前加保留字_____，表示该参数是常量参数，不加任何保留字表示该参数是_____。

3. 要想把子程序中对参数的赋值带回到主调子程序, 那么该参数应定义成_____。
4. 在子程序中不允许给_____参数赋值。
5. 下列函数的作用是求两个数的较大数, 并把较大数作为函数值返回, 请填空。

```
Function max_v(____):Real;
Var
    Max:Real;
begin
    max:=a;
    if (max<b) then
        max:=b;
    _____;
end;
```

三、程序设计题 (40 分, 每题 20 分)

1. 变量值的交换。程序设计界面如图 5-21 所示, 任意输入两个数给 a 和 b, 单击【判断】按钮, 如果 a 小于 b, 就交换它们的值, 否则不交换, 程序运行界面如图 5-22 所示。要求把交换两变量的值编写成一个过程。



图 5-21 程序设计界面

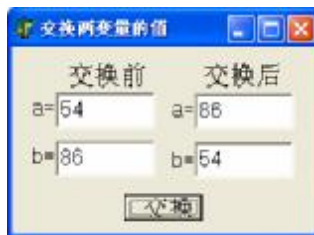


图 5-22 程序运行界面

2. 编写程序求表达式 $s = 1 - 1/(1 \times 2) + 1/(2 \times 3) - 1/(3 \times 4) + \dots + (-1)^{n-1} / ((n-1) \times n)$ 中 s 的值, 要求加到的那一项的值小于某一值 (精度) 时为止。要求用通用过程或函数来实现, 并且精度可以由用户任意决定。程序设计界面如图 5-23 所示, 程序运行时在精度后的编辑框中输入精度, 然后单击【计算】按钮, 则计算出 s, n 的值, 并在相应的编辑框中显示出来, 如图 5-24 所示。



图 5-23 程序设计界面



图 5-24 程序运行界面

第 6 章 用户自定义类型

本章要点

- ▮ 枚举类型的概念、定义与使用方法
 - ▮ 子界类型的概念、定义与使用方法
 - ▮ 集合类型的概念、定义与使用方法
 - ▮ 记录类型的概念、定义与使用方法
-

6.1 理论知识

在前面的章节中，已经介绍了 5 种标准数据类型，为了满足实际工作的需要，在 Delphi 中还引入了高级数据类型，即用户自定义类型，高级数据类型在使用前需要定义。本章将具体讲述枚举、子界、集合与记录等 4 种类型概念、定义与使用方法。

6.1.1 枚举类型的定义与使用

1. 枚举类型的定义

枚举类型就是将要用到的数据一一列举出来，因此，枚举类型要求数据个数有限，特别适合于表示物体的颜色、人的职业、星期几、月份等非数值型数据。枚举类型的定义格式如下。

[格式]:

Type

类型标识符=(标识符 1,标识符 2,标识符 3,⋯,标识符 n);

[功能]: 定义枚举类型，类型名由“类型标识符”指定。

[说明]: Type 表示类型定义段的开始;“类型标识符”为任意合法的标识符;“标识符 1”到“标识符 n”是枚举类型中的所有元素，它们又称为枚举常量。枚举常量必须是标识符，而且不能重复出现。

定义了枚举类型后，就可以声明枚举类型变量了，方法与其他类型变量的声明完全相同。

2. 枚举类型的使用

枚举类型属于顺序类型，枚举类型的每个元素对应一个有序的整数，其中第一个元素对应序数 0。因此，枚举类型本质上是用一些枚举常量来表示一组连续的整数。但枚举常量不能直接进行算术运算，可以进行关系运算或间接的算术运算。例如，有下列枚举类型的定义：


```
Type
Color1=(Red,Green,Blue,Yellow);
```

该语句定义了一个枚举类型，名称为 Color1，包含 4 个元素分别为：Red，Green，Blue，Yellow。其中，元素 Red 对应的序数为 0，Yellow 对应的序数为 3，即 Ord（Yellow）的值为 3，依此类推。每个枚举常量都有前趋值（第一个除外）和后继值（最后一个除外）。例如：Pred（Green）的值为 Red。另外，Low（Color1）和 High（Color1）分别表示 Color1 的第一个枚举常量 Red 和最后一个枚举常量 Yellow。由于每个枚举常量对应一个惟一的序数，因此它们可以进行比较。例如：Green<Yellow 的值为 True。

【例 6-1】 编写一个设置文本格式的应用程序，程序的设计界面如图 6-1 所示。程序运行时，在编辑框中显示的文本是“轻松学习 Delphi”。程序运行时，单击【宋体】按钮，文本字体为宋体；单击【隶书】按钮，文本字体为隶书；单击【黑体】按钮，文本字体变为黑体。图 6-2 是单击隶书时的运行界面。要求用枚举类型实现。



图 6-1 程序设计界面



图 6-2 程序运行界面

【实现分析】

文本字体一共只有 3 种，因此可以定义一个枚举类型和相应的枚举变量，枚举类型可定义为 MyFont=(st,ls,ht)。当枚举变量为“st”时，文本字体为“宋体”；当枚举变量为“ls”或“ht”时，文本字体为“隶书”或“黑体”。由于在 3 个按钮单击事件中都将用到枚举变量，因此将枚举变量定义为单元级变量。

【界面设计】

本例的组件属性设置及组件功能如表 6-1 所示。

表 6-1 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Button1	Caption	'宋体'	单击使文本字体变为宋体
Button2	Caption	'隶书'	单击使文本字体变为隶书
Button3	Caption	'黑体'	单击使文本字体变为黑体
Edit1	Text	'轻松学习 Delphi'	显示文本

【程序代码】

```
implementation
type
  MyFont=(st,ls,ht);           //定义枚举类型
var
```

```
ft:MyFont;                                //定义枚举类型变量
{$R *.dfm}
Function ffont(ff:MyFont):String;          //自定义函数
begin
    Case ff of
        st:ffont:='宋体';
        ls:ffont:='隶书';
        ht:ffont:='黑体';
    end;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Button1.Caption='宋体' then
    begin
        ft:=st;
        Edit1.Font.Name:=ffont(st);        //调用函数，使文本字体变为宋体
    end;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    if button2.Caption='隶书' then
    begin
        ft:=ls;
        Edit1.Font.Name:=ffont(ls);        //调用函数，使文本字体变为隶书
    end;
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    if Button3.Caption='黑体' then
    begin
        ft:=ht;
        Edit1.Font.Name:=ffont(ht);        //调用通用函数，使文本字体变为黑体
    end;
end;
end;
```

6.1.2 子界类型的定义与使用

1. 子界类型的定义

子界类型表示有序类型中的一部分，子界类型的定义格式及功能如下。

[格式]:

Type

类型标识符=常量 1..常量 2;

[功能]: 定义子界类型，类型名由“类型标识符”指定。

[说明]: “类型标识符”仍然是任意合法的标识符，名称由用户任意给定。“常量 1”和“常量 2”是子界类型的下界和上界，下界必须小于上界。注意“常量 1”和“常量 2”之间的符

号是“..”而非“...”。下界和上界及它们之间的所有元素都属于相同的顺序类型。由于子界类型只列举出了常量 1 和常量 2，因此系统和他人都应该知道下界和上界之间的所有元素，这在定义子界类型时需要注意。定义了子界类型后，同样可以进行子界类型变量的声明。另外，对子界类型变量的声明也可以像声明数组变量一样将定义与声明合二为一，下面的两种声明变量的方法其效果完全相同。

方法 1:

```
type
    Num=1..10;
var
    n1,n2:Num;
```

方法 2:

```
var
    n1,n2:1..10;
```

2. 子界类型的使用

子界类型允许进行多种运算，这取决于子界类型的基类型。和枚举类型一样，可以进行关系运算。不同之处在于，子界类型可以直接进行算术运算，而且子界类型的第一个元素的序数值为 1 而不是 0。

6.1.3 集合类型的定义与使用

1. 集合类型的定义

集合是具有相同性质但又可以区分开来的对象的全体。在数学与现实生活中，集合的对象可以是无限的，也可以是有限的。在 Delphi 中，集合的对象只能限定在某个范围之内，集合中的对象称为元素。与数组、枚举和子界类型不同，集合中的元素是没有先后顺序的。另外，集合中的元素是各不相同的，这与枚举和子界类型相似。集合类型的定义格式如下。

[格式]:

```
Type
    类型标识符=set of 基类型;
```

[功能]: 定义集合类型，类型名由“类型标识符”指定。

[说明]: Type 表示类型定义段的开始;“类型标识符”是合法的标识符名;这里的基类型只能是字符型、布尔型、枚举型和子界型 4 种顺序类型。集合中的所有元素类型相同。下面定义了两个集合类型，一个叫 Color，其基类型为枚举类型;另一个叫 Char，其基类型为子界类型。

```
Type
    Colors=(Red,Green,Blue,Yellow);           //先定义一个枚举类型
    Color=set of Colors;                       //定义一个基类型为枚举类型的集合类型
    Char=set of 'm'..'q';                     //直接定义一个基类型为子界类型的集合类型
```

注意，集合类型的元素个数不能超过 256，而且其序数值也必须在 0~255 范围之内。例如，下面的定义是错误的：

```
type
  Num_p=set of 200..400;      //虽然只有 201 一个元素，但 256~400 这些数值超过了 255
```

集合类型变量的声明与其他变量的声明完全相同，而且也有两种方法：先定义类型，再声明变量；或者声明变量与定义类型合二为一，如下所示：

```
var
  Char1:Set of 'm'..'q';
```

上面语句定义了一个基类型为子界类型的集合类型变量 Char1。

2. 集合类型的使用

定义了集合类型和声明了集合类型变量后就可以使用集合类型。在使用集合类型时需注意三点：集合变量的取值；集合的交、并和差运算；集合的各种关系运算。

对集合变量的取值，如果知道数学中集合的有关知识，就很容易理解。集合变量的取值是一个集合，称为集合值，它包括空集合在内的全体子集。例如，有 n 个元素的集合，其集合值有 2^n 个。另外还有两点需要注意：根据集合类型的定义，集合变量的取值与其元素的先后顺序无关；集合中的重复元素视作同一个元素。集合值的表示形式为：以方括号括起来的以逗号隔开的元素序列，例如 `['m','n','p']` 就是集合变量 Char1 的一个取值。

对于集合的交、并和差运算，需要注意两点：一是运算的适用条件——针对相同的集合类型；二是运算规则。

交运算的规则如下：集合 A 和集合 B 的交运算就是集合 A 和集合 B 的相同元素组成的集合，记作 $A*B$ 或者 $B*A$ 。例如：`[1,2,3]*[3,4,5]` 的值为 `[3]`。

并运算的规则如下：集合 A 与集合 B 的并运算就是集合 A 和集合 B 的所有元素组成的集合（重复的元素视作一个），记作 $A+B$ 或 $B+A$ 。例如：`[1,2,3]+[3,4,5]` 的值为 `[1,2,3,4,5]`。

差运算的规则如下：集合 A 与集合 B 的差运算就是所有属于集合 A 而不属于集合 B 的元素组成的集合，记作 $A-B$ 。例如 `[1,2,3]-[3,4,5]` 的值为 `[1,2]`。

对于集合的各种关系运算，主要应掌握集合之间的相等“=”、不等“ \neq ”、包含“ \supseteq ”和被包含“ \subseteq ”运算及数据与集合之间的属于“IN”运算。

当两种相同类型集合 A 和 B 的所有元素完全相同时，就说集合 A 等于集合 B，即 $A=B$ 或者 $B=A$ 的值为 True。当集合 A 中的所有元素完全包含于相同类型的集合 B 中时，就说集合 A 包含于（即被包含）集合 B，即 $A\subseteq B$ 的值为 True。反之，当集合 A 包含相同类型集合 B 中的所有元素时，就说集合 A 包含集合 B，即 $A\supseteq B$ 的值为 True。例如：`[2,3] \subseteq [1,2,3]` 的值为 True；而 `[2,3]=[1,2,3]` 的值为 False；`[2,3] \supseteq [1,2,3]` 的值也为 False。

属于运算“IN”是针对数据或元素对集合而言的。当数据 x 与集合 A 的基类型相同并且存在于集合 A 中时，就说数据 x 属于集合 A，即 $x \text{ IN } A$ 的值为 True；反之，其值为 False。

【例 6-2】 编写一个判断字符串中是否有元音字母的程序，程序的设计界面如图 6-3 所示。程序运行时，在【输入单词】编辑框中输入一个任意单词，然后单击【判断】按钮，将

在【结果】编辑框中显示有无元音字母的文本。其中，元音字母为“a,e,i,o,u”，程序运行界面如图 6-4 所示。要求用集合类型实现。



图 6-3 程序设计界面

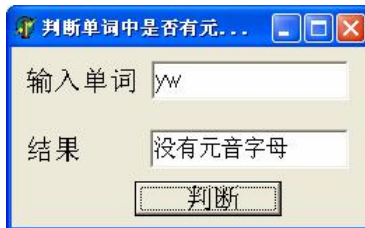


图 6-4 程序运行界面

【实现分析】

元音字母一共有 5 个：a，e，i，o，u。可以定义一个集合类型变量，例如 yyzmj，用来存放单词中的元音字母，设其初值为空集。利用循环语句逐个判断单词的每个字符，若是元音字母则向该集合中添加一个相应元素。最后判断 yyzmj 集合是否为空集，若不是空集说明该单词中有元音字母，否则该单词无元音字母。

【界面设计】

本例的组件属性设置及组件功能如表 6-2 所示。

表 6-2 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Button1	Caption	'判断'	单击可显示有无元音
Label1	Caption	'输入单词'	提示输入任意单词
Label2	Caption	'结果'	提示结果
Edit1	Text	''	用来输入单词
Edit2	Text ReadOnly	'' True	显示结果

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  yyzmj:set of Char;           //定义集合变量
  str:String;
  k,m:Integer;
begin
  yyzmj:=[];                   //给集合变量赋初值
  k:=Length(Edit1.Text);
  for m:=1 to k do
  begin
    str:=Copy(Edit1.Text,m,1);
    if str='a' then
      yyzmj:=yyzmj+'a';        //如有元音字母就给集合变量添加一个元素
    if str='e' then
```

```

        yyzmj:=yyzmj+'e';
    if str='i' then
        yyzmj:=yyzmj+'i';
    if str='o' then
        yyzmj:=yyzmj+'o';
    if str='u' then
        yyzmj:=yyzmj+'u';
    end;
    if yyzmj<>[] then //根据集合变量是否为空集显示相关文本
        Edit2.Text:='有元音字母'
    else
        Edit2.Text:='没有元音字母';
    end;
end;

```

6.1.4 记录类型的定义与使用

1. 记录类型的定义

在前面的章节中，已经介绍了数组、枚举、子界、集合共 4 种高级数据类型，这些类型的共同点是所属元素必须属于相同的类型。而现实生活中常需要把不同类型的数据组合在一起，例如最新的电子身份证，它包含很多信息：身份证号、姓名、出生日期、性别等，这些数据就不属于相同的类型，为了处理如电子身份证这样的由多种不同类型数据组成的实体，在 Delphi 中引入了记录类型。记录类型的定义格式与功能如下。

[格式]:

```

type
    记录类型标识符=Record
        字段 1:类型 1;
        字段 2:类型 2;
        ...
        字段 n:类型 n;
end;

```

[功能]: 定义记录类型，记录类型名由“记录类型标识符”指定。

[说明]: 保留字 **Type** 含义同前。“记录类型标识符”是任意合法的标识符，这条语句与定义枚举、子界和集合类型时的语句相似。下面是 n 个字段列表，又叫域名表，也是任意合法的标识符。而且这些标识符由于有记录类型限定，因而可以和记录类型外的标识符同名。“类型 1”到“类型 n ”是任意的标准类型或高级类型。最后是一条“end;”语句，这与本章的另外 3 种高级类型的定义格式是不同的。

下面定义一个住院病人的记录类型，如下所示：

```

type
    Patient=Record //说明这是记录类型
        P_num:Integer; //字段 1 是病人的住院号，为整型数据
        P_name:String; //字段 2 是病人的姓名，为字符串型数据
    end;

```

```
P_pay:Boolean;           //字段 3 是病人的付款情况，为布尔型数据
end;
```

定义了记录类型后，就可以声明记录变量了，声明记录型变量与声明其他类型变量基本相同。另外也可以将定义记录类型和声明记录变量合二为一，不过，记录类型与其他类型相比要复杂得多，建议最好将它们分开声明。

除了常规的记录类型以外，Delphi 还提供了变体型的记录类型，有兴趣可参考相关的资料，本章不再赘述。

2. 记录类型的使用

定义了记录类型和声明了记录类型变量后就可以使用记录类型了。使用记录类型的关键是如何访问记录变量的字段，有两种访问字段的方法：限定访问记录字段和使用 With 语句。

(1) 限定访问记录字段

格式与功能如下。

[格式]:

记录变量名.字段名

[功能]: 访问记录域。

[说明]: 对上面的格式可以这样理解，访问某个记录变量的某个字段。注意“.”是限定标识符，不能省略。

前面已经定义了一个住院病人的记录类型，请看下面的程序段：

```
var
    NewPatient:Patient;           //声明记录类型变量NewPatient
    NewPatient.P_num:=10112004;    //访问并给 NewPatient 变量的住院号字段赋值
    NewPatient.P_name:='ZhangSan'; //访问并给 NewPatient 变量的姓名字段赋值
```

在上述的语句中，如果记录变量有很多字段，在访问这些字段时都要书写记录变量名，这显然是比较烦琐的。为解决该问题，在 Delphi 中引入了 With 语句用来访问记录的字段。

(2) 使用 With 语句访问记录字段

其格式及功能如下。

[格式]:

With 记录变量名 Do 语句

[功能]: 用 With 打开“记录变量名”指定的变量，下面访问它的字段就不需再在域名前加“记录变量名”了。

[说明]: “语句”可以是简单语句也可以是复合语句（对于复合语句需要用 begin 和 end 括起来）。采用这种方法时，语句中字段名前的记录变量名和小圆点就省略不写了。例如：

```
with NewPatient Do
begin
    P_num:=10112004;           //省略字段名前的小圆点和记录变量名
    P_name:='ZhangSan';
end;
```

注意：With 语句也可以嵌套。

【例 6-3】 编写一个程序，用来输入和显示某单位员工基本信息。程序的设计界面如图

6-5 所示，程序运行时，单击【输入信息】按钮，用户可以输入员工的基本信息，包括姓名、职称和基本工资。单击【查看信息】按钮，可以显示刚刚输入的该单位员工的基本信息，如图 6-6 所示。

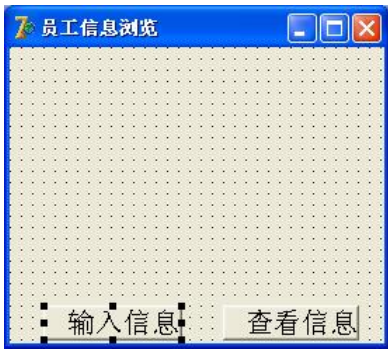


图 6-5 程序设计界面

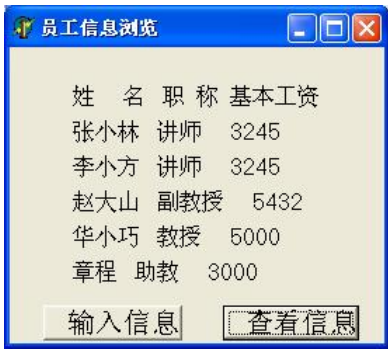


图 6-6 程序运行界面

【实现分析】

员工信息中的姓名和职称属于字符串类型，而基本工资属于整型，为记录员工信息，可定义一个记录类型（假设类型名为 `ygxx`）。为记录一个单位的多名员工的信息，可定义一个 `ygxx` 的数组，数组中的每个元素用来存放一个员工信息。输入时通过一个循环依次给每个数组元素的各个域输入数据，查看信息时可通过一个循环依次输出数组中的各个域的值。

【界面设计】

本例的组件属性设置及组件功能如表 6-3 所示。

表 6-3 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Button1	Caption	'输入信息'	单击输入员工信息
Button2	Caption	'查看信息'	单击显示员工信息
Form1	Caption	'员工信息浏览'	容纳其他组件

【程序代码】

```
implementation
type
  ygxx=Record                                //定义记录类型
    xm:String[8];
    zc:String[6];
    jbgz:Integer;
  end;
var
  yg:Array [1..5]of ygxx;                  //定义基类型为记录类型的数组变量

{$R *.dfm}
```



```

procedure TForm1.Button1Click(Sender: TObject);
var
  k:Integer;
begin
  for k:=1 to 5 do           //输入员工基本信息
  begin
    yg[k].xm:=InputBox('员工信息','请输入第'+IntToStr(k)+'个员工姓名,');
    yg[k].zc:=InputBox('员工信息','请输入第'+IntToStr(k)+'个员工职称,');
    yg[k].jbgz:=StrToInt(InputBox('员工信息','请输入第'+IntToStr(k)+'
                                '个员工基本工资,0'));

    end;
  end;

procedure TForm1.Button2Click(Sender: TObject);           //显示员工信息
var
  m:Integer;
begin
  Canvas.TextOut(45,25,'姓      名+'      '+'职  称+'      '+'基本工资');    //在指定位置显示
  for m:=1 to 5 do
    Canvas.TextOut(45,25*(m+1),yg[m].xm+'      '+'yg[m].zc+'
    '+'+IntToStr(yg[m].jbgz));
  end;

```

6.2 典型实例

6.2.1 典型实例一

【实例题目】：颜色选择程序

编写一个颜色选择程序，程序的设计界面如图 6-7 所示。程序运行时，单击【显示各种颜色】按钮，将显示各种颜色及代表该颜色的数字；在【你喜欢的颜色的数字=】编辑框中输入数字，单击【选择】按钮，将显示用户选择的颜色，程序运行界面如图 6-8 所示。

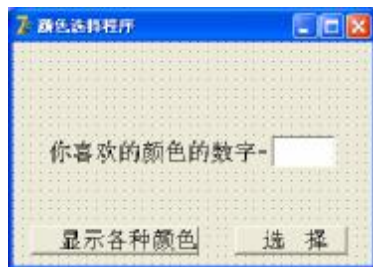


图 6-7 程序设计界面

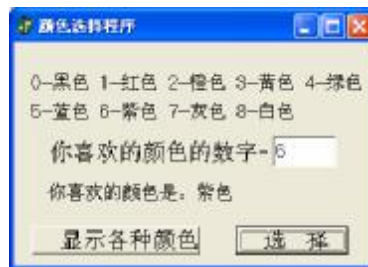


图 6-8 程序运行界面

【实现方法】

从图 6-8 可以看到, 本例共显示了 9 种颜色, 因此可以定义一个枚举类型 Color。注意, 枚举类型属于顺序类型, 其第一个元素的序数值为 0 而不是 1, 这与子界类型不同。

【界面设计】

本例的组件属性设置及组件功能如表 6-4 所示。

表 6-4 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'你喜欢的颜色的数字='	提示
Button1	Caption	'显示各种颜色'	单击它显示各种颜色及代表颜色的数字
Edit1	Text	"	用来输入要选择的颜色
Button2	Caption	'选 择'	单击它显示选择的颜色

【程序代码】

```
implementation
type
    Color=(Black,Red,Orange,Yellow,Green,Blue,Purple,Grey,White);    //定义枚举类型
var
    yc:Color;                    //定义枚举类型变量
{$R *.dfm}
Function FCol(fc:Color):String;    //自定义函数, 注意其位置
begin
    Case fc of                    //Case 多分支选择语句
        Black :FCol:='黑色';
        Red   :FCol:='红色';
        Orange:FCol:='橙色';
        Yellow:FCol:='黄色';
        Green :FCol:='绿色';
        Blue  :FCol:='蓝色';
        Purple:FCol:='紫色';
        Grey  :FCol:='灰色';
        White :FCol:='白色';
    end;
end;
procedure TForm1.Button1Click(Sender: TObject);    //显示各种颜色
begin
    Canvas.TextOut(15,25,'0--黑色 1--红色 2--橙色 3--黄色 4--绿色');
    Canvas.TextOut(15,50,'5--蓝色 6--紫色 7--灰色 8--白色');
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    m:Integer;
begin
```

```

m:=StrToInt(Edit1.Text);           //获取用户输入的颜色序号
Case m of                          //判断是何种颜色
  0:y:=Black;
  1:y:=Red;
  2:y:=Orange;
  3:y:=Yellow;
  4:y:=Green;
  5:y:=Blue;
  6:y:=Purple;
  7:y:=Grey;
  8:y:=White;
end;
Canvas.TextOut(30,120,'你喜欢的颜色是: '+FCol(y)); //调用函数并在指定位置显示
end;

```

6.2.2 典型实例二

【实例题目】：学生信息处理程序

假设某个班只有 5 名学生，考试完毕需要记录每个学生的学号、姓名和各科成绩（语文、数学、英语和文科综合）。编写一个学生信息的输入和显示程序，要求能够输入和显示学生的学号、姓名和各科成绩。程序设计界面如图 6-9，程序运行时，单击【输入并显示学生信息】按钮，将输入学生的信息并显示在窗体上，如图 6-10 所示。



图 6-9 程序设计界面

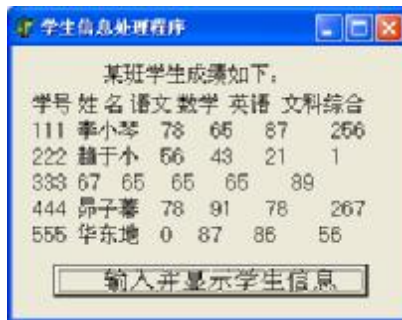


图 6-10 程序运行界面

【实现方法】

由于学生数据有几种不同类型的数据组成，为处理它，最好应定义一个记录型。为存放 5 名学生的信息，可定义一个记录型的数组，数组中的每个元素用来存放一名学生信息。

【界面设计】

在窗体上添加一个按钮，设置按钮的 Caption 属性值为“输入并显示学生信息”即可。

【程序代码】

```

implementation
const
  k=5;           //定义一个单元级常量

```

```

Type
    Student=Record           //定义记录类型
        num:Integer;
        name:String;
        yw,sx,yy,wz:Real;
    end;
var
    StArr:Array[1..k]of Student;    //定义基类型为 Student 的数组变量
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    m:Integer;
begin
    Canvas.TextOut(70,12,'某班学生成绩如下: ');
    for m:=1 to k do
        begin
            //以下输入学生的有关信息
            StArr[m].num:=StrToInt(inputbox('学生信息','请输入第'+IntToStr(m)
                +'个学生学号',IntToStr(m)));
            StArr[m].name:=inputbox('学生信息','请输入第'+IntToStr(m)+'个学生姓名,');
            StArr[m].yw:=StrToFloat(inputbox('学生成绩','请输入第'+IntToStr(m)+
                '个学生语文成绩','0'));
            StArr[m].sx:=StrToFloat(inputbox('学生成绩','请输入第'+IntToStr(m)+
                '个学生数学成绩','0'));
            StArr[m].yy:=StrToFloat(inputbox('学生成绩','请输入第'+IntToStr(m)+
                '个学生英语成绩','0'));
            StArr[m].wz:=StrToFloat(inputbox('学生成绩','请输入第'+IntToStr(m)+
                '个学生文科综合成绩','0'));
        end;
    for m:=1 to k do
        //以下显示学生的有关信息
        begin
            Canvas.TextOut(15,35,'学号 '+'姓 名 '+'语文 '+'数学 '+'英语 '+'文科综合');
            Canvas.TextOut(15,35+20*m,IntToStr(StArr[m].num)+' '+'StArr[m].name+'
                '+'FloatToStr(StArr[m].yw)+' '+'FloatToStr(StArr[m].sx)+'
                '+'FloatToStr(StArr[m].yy)+' '+'FloatToStr(StArr[m].wz));
        end;
    end;
end;

```

6.3 上机练习

6.3.1 上机练习一

【练习题目】：寻找 1~100 之间的全部素数

编写一个寻找 1~100 之间的全部素数的程序，程序的设计界面如图 6-11 所示。程序运行时，单击【寻找素数】按钮将显示 1~100 之间的全部素数，如图 6-12 所示。单击【清除】

按钮，窗体上显示的素数将消失。

【要点提示】

可用“筛选法”来求素数。该方法是著名的希腊数学家 Eratosthenes 提出来的。假设求 $1 \sim n$ 的全部素数，具体方法如下：首先将全部 n 个数放入一个筛中，去掉最小的数 1；用 2 去除它后面的每个数，把能被 2 整除的数挖掉；再用 3 去除它后面的每个数，同样把能被 3 整除的数挖掉；如此反复，直到除数为 \sqrt{n} （取其整数部分）；那么剩下的没被挖掉的数就是 $1 \sim n$ 之间的全部素数。可用集合来完成本问题，先把 $2 \sim 100$ 之间的所有数放到一个集合中，然后从该集合中筛去所有的非素数，剩下的就是素数。



图 6-11 程序设计界面

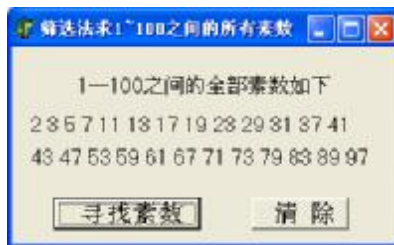


图 6-12 程序运行界面

【参考代码】

```
procedure TForm1.Button1Click(Sender: TObject);
Type
    aa=2..100;                //定义子界类型
    ssj=set of aa;            //定义集合类型，其基类型为子界类型
var
    ss:ssj;
    m,n,k:Integer;
    t:Real;
    str1,str2:String;
begin
    str1:="";
    str2:="";
    t:=Sqrt(100);              //求最大的除数
    k:=Trunc(t);
    ss:=[2..100];              //对集合变量赋值
    for m:=2 to k do
        for n:=m+1 to 100 do
            if (n mod m=0) then
                ss:=ss-[n];     //能被整除的数被挖掉
    Canvas.TextOut(50,20,'1---100 之间的全部素数如下');
    for m:=2 to 100 do
        if (m IN ss) then
            if (m<=41) then
```

```
begin
    str1:=str1+' '+IntToStr(m);
    Canvas.TextOut(10,50,str1);           //第一行显示的素数小于等于 41
end
else
begin
    str2:=str2+' '+IntToStr(m);
    Canvas.TextOut(10,75,str2);           //第二行显示的素数大于 41
end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Canvas.FillRect(Form1.ClientRect);     //清除窗体上显示的内容
end;
```

请思考这样一个问题，如果要求 1~300 之间的素数，还能用集合及“筛选法”来实现吗？

6.3.2 上机练习二

【练习题目】：摸彩球

商场搞促销活动，顾客凭购物发票摸彩球。在装球的小箱子里连续摸三次，每次摸的球都要放回去，如果三次摸的球颜色各不相同，那么该顾客就中奖了。假设箱子里一共只有三种颜色的球，分别为红色球、绿色球和蓝色球，请编写程序找出不同的摸奖办法，并显示摸出的球。程序设计界面如图 6-13 所示，程序运行界面如图 6-14 所示。要求用枚举类型实现。

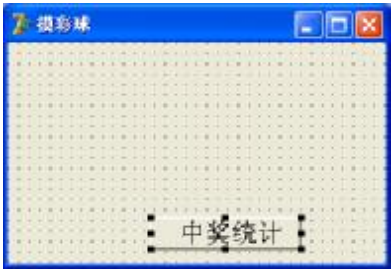


图 6-13 程序设计界面

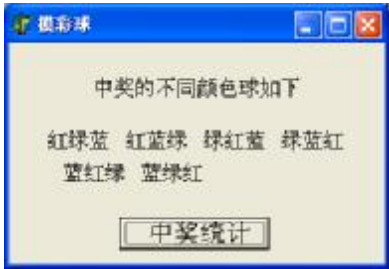


图 6-14 程序运行界面

【要点提示】

一共只有三种不同颜色的球，可以定义一个枚举类型变量，表示三种颜色的球。为了依次找到不同颜色的球，可以利用循环语句来实现。

【参考代码】

```
implementation
type
```

```

    zjj=(RBall,GBall,BBall);           //定义枚举类型
{$R *.dfm}
Function meth(zjfzjj):String;           //自定义函数
begin
    Case zjf of
        RBall:meth:='红';
        GBall:meth:='绿';
        BBall:meth:='蓝';
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    zj1,zj2,zj3zjj;                     //声明枚举变量
    str,str1,str2,str3:String;
begin
    str:="";
    for zj1:=Low(zjj) to High(zjj) do   //循环变量为枚举类型
    begin
        str1:=meth(zj1);
        for zj2:=Low(zjj) to High(zjj) do
        begin
            if (zj2<>zj1) then
            begin
                str2:=meth(zj2);
                for zj3:=Low(zjj) to High(zjj) do
                if ((zj3<>zj1)And(zj3<>zj2))then
                begin
                    str3:=meth(zj3);
                    str:=str+str1+str2+str3+'  ';    //找到一种不同颜色的球就添加上
                end;                                //此处 end 语句很多，注意不要漏掉或多写
                end;
            end;
        end;
    end;
    if Length(str)>33 then
    begin
        str1:=Copy(str,1,33);                //字符串太长就截取为两段
        str2:=Copy(str,34,Length(str));
        Canvas.TextOut(65,25,'中奖的不同颜色球如下');
        Canvas.TextOut(30,65,str1);
        Canvas.TextOut(30,90,str2);
    end;
end;

```

```
end;
end;
```

课后考场

一、选择题（20分，每题5分）

- 以下枚举类型定义正确的是_____。
 A. Type aa=('+', '-', '*', '/')
 B. Type aa=(+, -, *, /)
 C. Type a=(White, Orange, Green, Yellow)
 D. Type a=(2.3, 3.4, 4.5, 5.6)
- 下列子界类型定义正确的是_____。
 A. Type x=2.2..8.2
 B. Type x=3..433
 C. Type x='A'...'Z'
 D. Type x=a..q
- 下列集合类型定义正确的是_____。
 A. Type x=set of 100..260
 B. Type x=set of m.t
 C. Type x=set of 1.2..5.2
 D. Type x=set of 'a'..'d'
- 下列集合表达式中为空集的是_____。
 A. ['x', 'y']-['a', 'b', 'z']
 B. ['a'..'d']+['c', 'x']
 C. ['a'..'g']*['m'..'q']
 D. ['b', 'd']-['b', 'e']

二、填空题（40分，每空5分）

- 在 Delphi 语言中，有 5 种顺序类型，它们是_____。
- 本章学习了 4 种高级数据类型，其中_____类型的各元素可以是不同的数据类型，而其他三种类型的元素必须类型相同。
- 枚举类型、子界类型和数组类型的元素都与相对位置有关，其中，枚举类型的第一个元素的序数值为_____，子界类型的第一个元素的序数值为_____。
- 集合[2.5, 7, 9]与集合[2, 4, 7, 3, 5, 9]_____（填不等或相等）。
- 集合表达式['m', 'n', 'x']-['m', 'q']*['a'..'z']的值是_____。
- 有如下程序段：

```
Type
  Color=(Red, Green, Blue, Yellow);
Var
  Color1, Color2: Color;
```

假设 Color1:=Blue; Color2:=Red; 那么 Pred(Color1)的值为_____, Ord(Pred(Succ(Color2)))的值为_____。

三、程序设计题（40分，每题20分）

- 利用子界类型编写程序，输入 2000 年至 2099 年中的任意一年及其月份，判断输入的月份有多少天。程序设计界面如图 6-15 所示，程序运行界面如图 6-16 所示。



图 6-15 程序设计界面



图 6-16 程序运行界面

2. 利用记录类型编写程序, 统计家庭成员的收支情况。假设有 3 个家庭成员, 父亲和母亲都有收入, 子女没有收入, 但有支出。程序设计界面如图 6-17 所示, 程序运行界面如图 6-18 所示。



图 6-17 程序设计界面



图 6-18 程序运行界面

第 7 章 常用组件的使用

本章要点

- ▮ 文本类组件的使用方法
 - ▮ 按钮类组件的使用方法
 - ▮ 列表类组件的使用方法
 - ▮ 时钟组件和通用对话框组件的使用方法
 - ▮ 菜单、工具栏和状态栏的设计方法
-


7.1 理论知识

Delphi 7 提供了众多的组件，合理地应用这些组件，不但可以编写出功能强大的应用程序，而且还会使应用程序界面更加美观漂亮。本章将着重介绍这些组件的使用方法。

7.1.1 文本类组件的使用

文本类组件主要用于显示和输入文本，主要包括 TLabel（标签）组件、TEdit（编辑框）组件、TMaskEdit（掩码编辑框）组件和 TMemo（备注框）组件。

1. TLabel 组件

TLabel 组件位于 Standard 选项卡中，图标是，它的主要功能有两个：一是起说明作用（例如为没有文字说明的组件如 TEdit 等进行说明）；二是动态地输出一些程序的运行信息。一般不使用 TLabel 组件的方法和事件，下面只介绍它的常用属性。

- ▮ Caption 属性：即标题属性，是 TLabel 组件最重要的属性，用来设置该组件中显示的文本。
- ▮ AutoSize 属性：用于决定标签是否自动随文本的长短而改变大小。当为 True 时可以改变大小，为 False 时不可改变大小。
- ▮ WordWrap 属性：用于决定标签的文本是否可以折行显示。当为 True 时表示可以折行显示，为 False 时表示不可以折行显示。
- ▮ Alignment 属性：用于决定标签中显示的文本的对齐方式，属性值有 taLeftJustify（左对齐）、taCenter（居中）和 taRightJustify（右对齐）。
- ▮ Enabled 属性：用于决定该组件是否能响应用户的操作。当其值为 True 时，该组件能响应用户的操作；为 False 时，组件变灰显示不响应用户的操作。一般组件均有该属性。
- ▮ Visible 属性：用于决定在程序运行时组件是否可见，当其值为 True 时，程序运行时组

件可见, 当其值为 **False** 时, 程序运行时组件不可见。

- ! **Font** 属性: 用于设置标签组件中显示的文本字体, 是一个对象属性, 具有 **Color** (字体颜色)、**Name** (字体名)、**Size** (字体大小) 等子属性。

2. TEdit 组件

TEdit 组件位于 **Standard** 选项卡中, 图标是 , 主要用来输入或输出单行文本, 下面介绍它的常用属性、方法和事件。

(1) TEdit 组件的常用属性

- ! **Text** 属性: 该属性代表显示在编辑框中的文本。程序运行时, 可以编辑修改该属性内容, 用户输入的文本都保存在 **Text** 属性中。
- ! **ReadOnly** 属性: 用于决定编辑框中显示的文本是否可以修改。属性值为 **True** 时不能修改, 属性值为 **False** 时能修改。
- ! **PasswordChar** 属性: 用于决定用户输入的字符是原样显示还是以密码方式显示。默认属性为 “#0”, 用户输入的字符将原样显示。当该属性为其他字符时, 用户输入的一切字符将以该属性中的字符显示出来, 即以密码方式显示。
- ! **CharCase** 属性: 用于控制编辑框中文本的大小写, 取值有三种情况, 分别如下: **ecNormal** (正常显示)、**ecLowerCase** (把所有的字母均显示成小写字母)、**ecUpperCase** (把所有的字母均显示成大写字母)。
- ! **AutoSelect** 属性: 用于决定当编辑框获得焦点时, 是否自动选定显示的文本。值为 **True** 时, 将自动选定, 值为 **False** 时, 将不会自动选定里面的文本。
- ! **BorderStyle** 属性: 用于决定编辑框组件是否有边框, 取值有两种情况: **bsSingle** (单线边框) 和 **bsNone** (无边框), 默认为无边框。
- ! **CanUndo** 属性: 用于指出对编辑框中显示的内容改变是否能够撤销。值为 **True** 时, 表示可以撤销, 值为 **False** 时, 表示不能撤销。
- ! **HideSelection** 属性: 用于决定当编辑框失去焦点时, 选中的文本是否还是以选中的方式提供视觉提示, 值为 **False** 时, 依旧提供视觉提示, 值为 **True** 时, 不再提供视觉提示。
- ! **MaxLength** 属性: 用于决定编辑框中最多能够输入或显示的字符个数。
- ! **Modified** 属性: 是一个运行属性, 用于指示编辑框中的内容是否发生了改变, 值为 **True** 时, 表示发生了改变, 值为 **False** 时, 表示没有发生改变。
- ! **SelStart** 属性: 是一个运行属性, 用于设置或指示选中文本的第一个字符的位置, 文本框的第一个字符位置为 0。如果没有选中文本, 该属性的值指示插入点所在的位置。如果在程序中设置了该属性的值, 将取消原来所做的选定。
- ! **SelText** 属性: 是一个运行属性, 用于设置或指示选中的文本。如果有选中的文本, 给该属性赋值, 赋的值将替换掉选中的文本。如果没有选中的文本, 给该属性赋值, 赋的值将插入到插入点位置处。
- ! **SelLength** 属性: 是一个运行属性, 用于设置或指示选中文本的长度。在选中文本时, 该属性值自动被设置。

(2) TEdit 组件的常用方法

- ! **SetFocus** 方法: 用于为编辑框设置焦点, 无参数。

- | **Clear** 方法：用于清除编辑框中的所有文本，无参数。
- | **ClearSelection** 方法：用于清除编辑框中选中的文本，无参数。
- | **Undo** 方法：用于撤销在编辑框中最近所做的编辑操作。
- | **ClearUndo** 方法：用于清除“撤销”缓冲区，从而不能对以前的操作做撤销操作，该方法无参数。
- | **SelectAll** 方法：用于选中编辑框中的所有文本。
- | **CopyToClipboard** 方法：用于把选中的文本复制到剪贴板上，相当于复制操作。
- | **CutToClipboard** 方法：用于把选中的文本移动到剪贴板上，相当于剪切操作。
- | **PasteFromClipboard** 方法：用于把剪贴板上的文本粘贴到插入点处，相当于粘贴操作。如果有选中的文本，选中的文本将被替换。

(3) TEdit 组件的常用事件

- | **OnChange** 事件：当 **Text** 属性值发生改变时将触发该事件。
- | **OnKeyPress** 事件：当按下键盘上一个 ASCII 码键时将触发该事件。注意，非 ASCII 码键不会触发该事件，这是与下面将要学习的 **OnKeyDown** 事件最重要的区别。该事件的语法如下：

```
type TKeyPressEvent = procedure (Sender: TObject; var Key: Char) of object;
```

其中变参 **Key** 用来记录所按下的键。

- | **OnKeyDown** 事件：当按下键盘上的任何键时都会触发该事件。该事件的语法格式如下：

```
TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;
```

其中，变参 **Key** 用来指出所按的键盘键，如果是非字母键，则代表的是虚拟键代码，虚拟键代码请参见附录 A。**Shift** 参数用来指出是否按了 **Shift**、**Alt** 和 **Ctrl** 键。

- | **OnKeyUp** 事件：当按下任何一个键后再松开时将触发该事件。该事件的语法格式如下：

```
procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;
```

各参数的含义与 **OnKeyDown** 事件完全一致。

【例 7-1】 设计一个简易账号和密码的检验程序。对输入的账号和密码规定如下：

(1) 账号为不超过 6 位的数字，密码为 4 位字符，在本例中，账号假设为 123456，密码假定为 Pass。

(2) 输入密码时，在屏幕上不显示输入的字符，而用“*”代替。

(3) 当输入不正确，如账号为非数字字符或账号和密码输入不正确时，将显示出消息框进行提示。程序的设计界面如图 7-1 所示，程序的运行界面如图 7-2 所示。

【实现分析】

要使账号不超过 6 位数字，可将编辑框的 **MaxLength** 属性设置为 6。在输入账号时，当输入的字符不是数字字符时，应给出提示并清除输入的字符，实现方法是在输入每一个字符时将触发 **OnKeyPress** 事件，该事件有一个参数 **Key**，通过对该参数的判断就可知道输入的是否为数字。输入完账户信息后按 **Enter** 键时，应判断账号是否正确，若不正确则给出提示信息，若正确则把焦点移到密码输入编辑框中，是否按的是 **Enter** 键也可以在 **OnKeyPress** 事件中进行判断。密码为 4 个字符，要使在输入密码的文本框中输入的每一个字符均显示为“*”，只

要将该编辑框的 MaxLength 属性值设置为 4、PasswordChar 属性设置为 “*” 即可。当输入结束时，按 Enter 键，在 OnKeyPress 事件中判断输入的是否是 Enter 键，如果是则判断密码输入是否正确，如果不正确则给出错误提示信息，如果正确则显示“欢迎进入系统”的提示文字。



图 7-1 程序设计界面

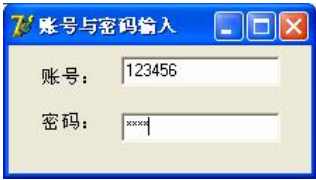


图 7-2 程序运行界面

【界面设计】

窗体组件属性设置及其作用如表 7-1 所示。

表 7-1 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'账号'	
Edit1	MaxLength	6	输入账号
Label2	Caption	'密码'	
Edit2	MaxLength PasswordChar	4 '*'	输入密码

【程序代码】


```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if ord(key)=13 then                                //如果是 Enter 键
        if edit1.text<>'123456' then                    //如果账号不正确
            begin
                ShowMessage('账号有错误，请重新输入'); //显示提示信息
                edit1.text:='';
                edit1.setFocus;
            end
        else
            Edit2.SetFocus                               //把焦点移至 Edit2，以便输入密码
    else
        if (key<'0') or (key>'9') then                  //如果输入的不是数字
            begin
                Showmessage('账号必须为数字');          //显示提示信息
                key:=#0;                                  //清除输入的字符
            end;
        end;
end;
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
```

```

if ord(key)=13 then                                //如果是 Enter 键
  if edit2.text='Pass' then                        //如果密码正确
    begin
      ShowMessage('欢迎你使用本系统! ');          //进入系统提示
      Form1.Close ;                               //关闭窗口体
    end
  else                                             //密码输入不正确
    begin
      ShowMessage('密码错误, 请重新输入! ');      //提示密码错
      Edit2.text:="";
      edit2.SetFocus ;
    end;
  end;
end;

```

3. TMemo 组件

TMemo 组件（又称备注）位于【Standard】选项卡下，图标是，它的主要作用是处理多行文本，可用于编辑文件。TMemo 组件具有 TEdit 组件所具有的全部属性、方法和事件。除此以外它还具有自己一些有特色的属性和方法，下面将一一加以介绍。

（1）TMemo 组件的常用属性

- **ScrollBars** 属性：用于设置备注框是否出现滚动条以及滚动条的种类。取值有 **ssNone**（无滚动条）、**ssBoth**（既有水平滚动条也有垂直滚动条）、**ssHorizontal**（水平滚动条）、**ssVertical**（垂直滚动条）4 种情况。
- **Lines** 属性：按行存放文本属性。其中，**Lines** 属性属于 **TStrings** 类，因此可以调用该类的相应方法灵活处理 **Lines** 中的数据。**Lines** 属性也具有一些属性，如它的一个常用属性 **Count**，用来表示 **Lines** 中有多少个字符串，即行数。
- **WordWrap** 属性：用于决定当输入的文本到达右边界时是否插入软回车，以便换到下一行再输入。

（2）TMemo 组件的常用方法

TMemo 组件的方法基本与 TEdit 组件一致，其中它的 **Lines** 属性属于 **TStrings** 类，它是一个字符串列表，该属性有一系列的方法需要掌握。

- **Add** 方法：用于向字符串列表的尾部添加一个字符串。其语法格式如下：

```
function Add(const S: string): Integer; virtual;
```

其中，参数 **s** 为要添加到字符串列表尾部的字符串。

- **Clear** 方法：用于清空字符串列表。其语法格式如下：

```
procedure Clear; virtual; abstract;
```

- **Delete** 方法：用于从字符串列表中删除一个指定的字符串。其语法格式如下：

```
procedure Delete(Index: Integer); virtual; abstract;
```

其中，参数 **Index** 是要删除的字符串的序号，字符串列表中的第一个字符串的序号为 0。

- **Insert** 方法：用于向字符串列表中插入一个字符串。其语法格式如下：

```
procedure Insert(Index: Integer; const S: string); virtual; abstract;
```

其中，参数 **Integer** 代表要插入字符串的位置序号，参数 **S** 为要插入的字符串。

▮ **Move** 方法：用于在字符串列表中移动字符串的位置。其语法格式如下：

```
procedure Move(CurIndex, NewIndex: Integer); virtual;
```

其中，参数 **CurIndex** 表示要移动的字符串的原来位置序号，参数 **NewIndex** 代表字符串移动后的位置序号。

▮ **LoadFromFile** 方法：用于指定的文件填充字符串列表。其语法格式如下：

```
procedure LoadFromFile(const FileName: string); virtual;
```

其中，参数 **FileName** 代表要填充到列表中的文件名。

▮ **SaveToFile** 方法：用于把字符串列表中的数据写到某文件中。其语法格式如下：

```
procedure SaveToFile(const FileName: string); virtual;
```

其中，参数 **FileName** 代表要写入的文件名。

【例 7-2】 编写一个对文本文件进行操作的应用程序，程序的设计界面如图 7-3 所示。程序运行时在【输入文件名】文本框中输入一个文本文件名，然后单击【打开】按钮将打开该文件。当修改了文件的内容后，可以通过单击【保存】按钮把文件保存起来。【剪切】、【复制】和【粘贴】按钮的功能与剪贴板的相应命令功能一致。【删除】按钮的功能是删除选中的文本。程序的运行界面如图 7-4 所示。



图 7-3 程序设计界面



图 7-4 程序运行界面

【实现分析】

打开文件可使用 **TMemo** 组件 **Lines** 属性的 **LoadFromFile** 方法，保存文件可使用 **TMemo** 组件 **Lines** 属性的 **SaveToFile** 方法，剪切操作可使用 **TMemo** 组件的 **CutToClipboard** 方法，复制操作可使用 **TMemo** 组件的 **CopyToClipboard** 方法，粘贴操作可使用 **TMemo** 组件的 **PasteFromClipboard** 方法，删除操作可使用 **TMemo** 组件的 **ClearSelection** 方法。需要注意的是，只有 **TMemo** 组件中的内容发生了变化时才能保存文件，只有选中了文本后才能进行剪切、复制、删除操作，执行了剪切或复制命令后才能执行粘贴操作。

【界面设计】

窗体组件属性设置及其作用如表 7-2 所示。

表 7-2 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'输入文件名'	提示输入文件名
Edit1	Text	"	输入要打开的文件名
Memo1	Lines	"	显示并编辑文件
Button1	Caption	'打开'	单击它将打开文件
Button2	Caption	'保存'	单击它将保存文件
Button3	Caption	'剪切'	单击它将执行剪切操作
Button4	Caption	'复制'	单击它将执行复制操作
Button5	Caption	'粘贴'	单击它将执行粘贴操作
Button6	Caption	'删除'	单击它将删除选中的文本

【程序代码】

```
var
    Form1: TForm1;
    FName:String;                                //存放打开的文件名
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);    //打开文件
begin
    FName:=Edit1.Text ;                          //获取要打开的文件名
    Memo1.Lines.LoadFromFile(FName);               //打开该文件
end;
procedure TForm1.Memo1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    if Memo1.SelLength >0 then                    //如果有文本被选中
    begin
        button3.Enabled :=true;                   //剪切按钮可用
        button4.Enabled:=true;                     //复制按钮可用
        button6.Enabled:=true;                     //删除按钮可用
    end
    else
    begin
        button3.Enabled :=false;                   //剪切按钮不可用
        button4.Enabled:=false;                     //复制按钮不可用
        button6.Enabled:=false;                     //删除按钮不可用
    end
end;
end;
procedure TForm1.Memo1Change(Sender: TObject);    //Memo 组件的内容发生了改变
begin
```




```

    Button2.Enabled :=true;           //保存按钮可用
end;
procedure TForm1.Button2Click(Sender: TObject); //保存按钮
begin
    Memo1.Lines.SaveToFile(FName);     //保存文件
    Button2.Enabled :=False;          //保存按钮不可用
end;
procedure TForm1.Button3Click(Sender: TObject); //剪切按钮
begin
    Memo1.CutToClipboard;             //执行剪切操作
    Button5.Enabled :=True;           //粘贴按钮可用
    button3.Enabled :=false;          //剪切按钮不可用
    button4.Enabled:=false;           //复制按钮不可用
end;
procedure TForm1.Button4Click(Sender: TObject); //复制按钮
begin
    Memo1.CopyToClipboard ;           //执行复制操作
    Button5.Enabled :=True;           //粘贴按钮可用
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
    Memo1.PasteFromClipboard ;         //粘贴操作
    Button3.Enabled:=False;            //剪切按钮不可用
    Button4.Enabled :=False;           //复制按钮不可用
end;
procedure TForm1.Button6Click(Sender: TObject); //删除按钮
begin
    Memo1.ClearSelection ;             //删除选中的文本
end;

```

4. TMaskEdit 组件

TMaskEdit 组件位于【Additional】选项卡下，图标是。它是一种格式化的编辑框，要求必须按照指定的格式输入数据。TMaskEdit 组件具有 TEdit 组件所具有的全部属性、方法和事件，另外还具有自身的一些特殊属性。

- EditMask 属性：用于控制输入数据的格式。可以用 Delphi 的【Input Mask Editor】对话框来设置 EditMask 属性，当然也可以选择某个样本来输入掩码字符串。
- EditText 属性：该属性代表 TMaskEdit 组件中输入的格式化文本。对于每个没有输入的字符用空白字符代替，如果输入了字符后，输入的字符将取代空白字符。

使用 TMaskEdit 组件的关键在于弄清掩码字符串 EditMask 的具体含义。掩码字符串 EditMask 属性分三个部分，每部分以分号隔开。第一部分确定数据的格式，是最主要的部分，也是最烦琐的部分，该部分中将用到的字符及其含义如表 7-3 所示。第二部分决定是否将掩码中的非用户输入数据和标准分隔符作为数据的一部分保存，只有 1 和 0 两种选择，取值为 1 表示作为数据的一部分保存，取值为 0 表示不作为数据的一部分保存。第三部分指出在掩码中未输入的数据（空格）用什么字符代替。


表 7-3 掩码字符串第一部分的特殊字符及其含义

字 符	含 义
!	如果出现该符号，则输入字符串的前导空格不会存成数据；否则，输入字符串的末尾空格不会存成数据
>	如果出现该符号，则它后面的所有字符都变为大写，除非遇到“<”
<	如果出现该符号，则它后面的所有字符变为小写，除非遇到“>”
<>	如果出现该符号，则它后面的所有字符大小写状态不变
\	“\”后面的字符依原样显示
/	用来分隔日期中的年月日
L	表示在该字符位置只能输入一个字母
l	表示在该字符位置只能输入一个字母，但可以输入不
A	表示在该字符位置只能输入一个字母或数字
a	表示在该字符位置只能输入一个字母或数字，但可以输入不
C	表示在该字符位置可以输入任意一个字符
c	表示在该字符位置可以输入任意一个字符，但可以输入不
9	表示在该位置只能输入一个数字字符，但可以输入不
0	表示在该位置只能输入一个数字字符
#	表示在该位置只能输入一个数字字符或正负符号，但不一定输入
:	用来分隔日期中的时分秒
;	用来分割掩码字符串中的三个部分
_	自动在 Text 中插入一个空格，在用户输入时，光标会自动跳过该字符

7.1.2 按钮类组件的使用

按钮类组件的使用非常普遍，在程序中主要用于执行命令，主要包括 TButton（按钮）组件、TBitBtn（位图按钮）组件、TCheckBox（复选框）组件、TRadioButton（单选按钮）组件和 TRadioGroup（单选按钮组）组件。

1. TButton 组件

几乎每个程序都要用到 TButton 组件，该组件位于【Standard】选项卡下，图标是，它的主要功能是响应鼠标的单击事件，并执行相应的命令。

（1）TButton 组件的主要属性


- ! Caption 属性：标题属性，即按钮显示的文本。
- ! Cancel 属性：用于决定该按钮是否为取消按钮，默认值为 False。当为 True 时，按 Esc 键就相当于单击了该按钮组件。
- ! Default 属性：用于决定该按钮是否为默认按钮，默认值为 False。当为 True 时，按 Enter 键就相当于单击了该按钮。注意，根据约定俗成的编程习惯，通常只把 OK、Yes 按钮的 Default 属性设置成 True，把 Cancel、No 按钮的 Cancel 属性设置成 True，至于其他按钮，这两个属性就使用默认值。

（2）TButton 组件的常用事件

- ! OnClick 事件：单击事件，是 TButton 组件的最常用事件。用鼠标单击 TButton 组件或 TButton 组件获得焦点时按 Enter 键或空格键时触发该事件。

- l OnMouseDown 事件：鼠标按下瞬间触发该事件。
- l OnMouseMove 事件：鼠标在 TButton 组件上移动时触发该事件。
- l OnMouseUp 事件：在 TButton 组件上松开按下的鼠标时将触发该事件。

2. TBitBtn 组件

TBitBtn 组件与 TButton 组件相似，不同之处是该组件可以显示一个彩色的位图，让人更容易理解。该组件位于【Additional】选项卡下，图标是。该组件的属性与响应的事件基本同 TButton 组件，下面仅介绍一下该组件的特殊属性。



- l Glyph 属性：用于为按钮指定一个位图文件，显示在按钮的表面。
- l Kind 属性：用于决定位图按钮的种类。位图按钮使用最多的属性就是 Kind。图 7-5 是当该 Kind 属性取不同值时位图按钮的形状。



图 7-5 各种不同 Kind 属性值时的位图按钮

注意：Close 按钮（Kind 属性为 bkClose）包含了一个关闭窗体的命令，因此它具有自动关闭窗体的功能。

3. TRadioButton 组件和 TRadioGroup 组件

TRadioButton 又称单选按钮，位于【Standard】选项卡中，图标是。单选按钮通常成组出现，用于为用户提供两个或多个互斥选项，即在一组单选按钮中只能选择一个。单选按钮一般是作为一个组来工作，这就是单选按钮组组件 TRadioGroup，该组件也位于【Standard】选项卡中，图标为。单选按钮组中的单选按钮是相互排斥的，即任何时候只能选中一个单选按钮，选中某单选按钮后同组的其他单选按钮自动变为未选中状态。注意，在程序设计时，如果要向单选按钮组添加单选按钮，只能通过编辑 Items 属性。出现在 Items 中的每行字符串将成为一个单选按钮。

（1）TRadioButton 组件的主要属性

- l Caption 属性：用于设置单选按钮的提示文字。
- l Alignment 属性：用于决定单选按钮的标题文字出现的位置，取值为 taLeftJustify 时，标题文字将出现在小圆圈左边，取值为 taRightJustify 时，标题文字将出现在小圆圈右边。默认为右边。
- l Checked 属性：用于标记单选按钮的状态，值为 True 时，单选按钮被选中，值为 False 时，单选按钮未被选中。

（2）TRadioButton 组件的常用事件

- l OnClick 事件：在单选按钮上单击时将发生该单选按钮的 OnClick 事件。

（3）TRadioGroup 组件的主要属性

- l Caption 属性：单选按钮组的标题属性，通常用于提示该单选按钮组的作用。

- ! **Items** 属性：用于设置单选按钮组中的单选按钮的标题，是一个字符串列表类型。
- ! **ItemIndex** 属性：用于设置单选按钮组中被选中按钮的序号，第一个按钮的序号为 0。
- ! **Columns** 属性：用于设置单选按钮组中单选按钮排列的列数，默认值为 1。

(4) TRadioGroup 组件的常用事件

- ! **OnClick** 事件：单击事件，在单选按钮组上单击时发生。

注意：也可以用 TGroupBox 组件来给单选按钮分组，位于一个 TGroupBox 组件中的多个单选按钮可看成一组，并可通过 TGroupBox 组件的 Caption 属性来提示这组单选按钮的作用。

4. TCheckBox 组件

TCheckBox 组件又称复选框，位于【Standard】选项卡下，图标是 。复选框与单选按钮类似，也提供一组选项供用户选择。但每个复选框都是一个单独的选项，不存在互斥的问题，可以从一组复选框中同时选择一项或多项，甚至不选。

(1) TCheckBox 组件的主要属性

- ! **Checked** 属性：用于标志复选框组件是否被选中，值为 True 时表示选中，值为 False 时表示未被选中。
- ! **AllowGrayed** 属性：用于设定复选框是否能处于变灰状态。如果该值为 False，表示复选框只能有两种状态——选中 and 未被选中，如果该值为 True，则表示复选框有三种状态——选中、未选中 and 变灰。
- ! **State** 属性：用于确定该组件的状态，取值有三种情况：cbUnchecked（表示未被选中）、cbChecked（表示被选中）、cbGrayed（表示变灰）。

(2) TCheckBox 组件的常用事件

- ! **OnClick** 事件：单击事件，在复选框上单击时将发生该事件。

【例 7-3】 程序运行时，在单选按钮组“统计年龄”中将自动生成 3 行 2 列共 5 个单选按钮，如图 7-7 所示。在统计年龄单选按钮组中任意选中一个单选按钮，备注框中立即显示所选的项。单击【关闭】按钮或【Close】位图按钮，程序关闭。单击【不可用】按钮，统计年龄单选按钮组不可用。程序设计界面如图 7-6 所示，程序运行界面如图 7-7 所示。



图 7-6 程序设计界面



图 7-7 程序运行界面

【实现分析】

为了在程序运行时，单选按钮组自动生成 5 个项目，在窗体的 OnCreate 事件中利用 RadioGroup1.Items.Add 方法生成 5 个项目即可。为了使单选按钮以 3 行 2 列显示，可以在对象观察器中设置 RadioGroup1 的 Columns 为 2。在选择某个单选按钮组时，为使备注框立即显示该项目，可以利用单选按钮组的 ItemIndex 属性（该属性返回选中的单选按钮的序号）识

别选中的单选按钮，再利用备注框的 Memo1.Lines.Add 方法把所选的单选钮的 Caption 属性添加进来。为了使单选按钮组不可用，只要把单选按钮组的 Enabled 属性设置为 False 即可。

【界面设计】

窗体组件属性设置及其作用如表 7-4 所示。

表 7-4 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Button1	Caption	'关闭'	单击关闭程序
Button2	Caption	'不可用'	单击使单选按钮组不可用
BitBtn1	Kind	bkClose	关闭窗口体
Memo1			显示有关信息
RadioGroup1	Caption Columns	'统计年龄' 2	生成各选项


【程序代码】

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    RadioGroup1.Items.Add('1-19 岁');           //添加单选按钮
    RadioGroup1.Items.Add('20-39 岁');
    RadioGroup1.Items.Add('40-59 岁');
    RadioGroup1.Items.Add('60-79 岁');
    RadioGroup1.Items.Add('80-99 岁');
end;
procedure TForm1.RadioGroup1Click(Sender: TObject);
//把选中的单选按钮的标题显示在 Memo 组件中
begin
    Memo1.Clear ;                               //清除 Memo 组件中的内容
    Memo1.Lines.Add('你属于'+RadioGroup1.Items[RadioGroup1.ItemIndex]+'组');
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Application.Terminate;                       //关闭程序
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    RadioGroup1.Enabled:=False;                  //让单选按钮组不可用
end;
```

7.1.3 列表类组件的使用

列表类组件通常用于从一组给定的选项选择一个或多个选项，包括 TListBox 组件和 TComboBox 组件。

1. TListBox 组件

TListBox 组件又称列表框，位于【Standard】选项卡下，图标是，它显示一个选项列表供用户选择。在列表框中，用户一次可以选择一项，也可以选择多项。

(1) TListBox 组件的常用属性

- l **Items** 属性：用于存放列表框中的列表项，属于 TStrings 类。
- l **Columns** 属性：用于设置在列表框中显示列表项的列数。
- l **Count** 属性：一个运行属性，用于返回列表框中列表项的数目。
- l **MultiSelect** 属性：用于设置能否在列表框中选择多个列表项，即是否允许多选。值为 True 时，允许多选，即可以选中多个列表项；值为 False 时，不允许多选，即只能选择一个列表项。
- l **ExtendedSelect** 属性：该属性只有在 MultiSelect 属性值为 True 时才起作用，用于决定列表框是否允许扩展多选。该属性值为 True 时表示允许扩展多选，此时选中一个选项后，按住 Shift 键再单击另一个选项，就把这两个选项间的所有选项均选中，实现连续选定。按住 Ctrl 键，再一一单击其他的选项，将把单击的选项选中，从而实现不连续选定。该属性值为 False 时，鼠标单击或按空格键将选择当前列表项或撤销对当前列表项的选定。
- l **ItemIndex** 属性：用于设置或返回选中的列表项的序号，第一个列表项的序号值为 0。在 MultiSelect 属性值为 False 时，给该属性赋某一系列项序号将选中对应的列表项，选中某列表项，该属性将返回选中列表项的序号，如果没有列表项被选中，该属性值为-1。在 MultiSelect 属性值为 True 时，该属性的默认值为-1，当选中多个列表项时，该属性代表选中的列表项中最近被选中的列表项的序号。
- l **Selected** 属性：用于测试某个列表项是否被选中，其基本使用方法为 Selected[Index]，其中 Index 代表列表项的序号，如果该属性值为 True，则序号为 Index 的列表项被选中，如果该属性值为 False，则序号为 Index 的列表项未被选中。
- l **SelCount** 属性：在 MultiSelect 属性值为 True 时，用于返回选中的列表项的个数。如果 MultiSelect 属性值为 False，则该属性值总是-1。
- l **Sorted** 属性：用于设置列表框中的列表项是否按字母顺序排序，值为 True 时，列表项将按字母顺序排序，值为 False 时，将不按字母顺序排序，即按照列表项的加入顺序显示。
- l **TopIndex** 属性：用于设置或获取显示在列表框中的第一个列表项的序号。

(2) TListBox 组件的常用方法

- l **Clear** 方法：用于清除列表框中的所有列表项，该方法无参数。
- l **ClearSelection** 方法：用于清除列表框中所有被选中的选项。
- l **SelectAll** 方法：用于选中列表框中的所有文本。
- l **MoveSelection** 方法：用于把列表框中的所有选中的列表项移动到另一个列表框中。其语法格式如下：

```
procedure MoveSelection(Destination: TCustomListControl); virtual;
```

其中参数 Destination 代表目标列表框。

- 1 CopySelection 方法：用于把列表框中的所有选中的列表项复制到另一个列表框中。其语法格式如下：

```
procedure CopySelection (Destination: TCustomListControl); override;
```


其中参数 Destination 代表目标列表框。

另外还可以使用列表框的 Items 属性来对列表项进行添加、插入、删除、移动等操作，此处不再赘述。

(3) TListBox 组件的常用事件

TListBox 组件响应的主要事件有：OnClick、OnKeyDown、OnKeyPress、OnKeyUp 等。

2. TComboBox 组件

TComboBox 组件又称组合框，位于【Standard】选项卡中，图标为。组合框分两个部分显示：顶部是一个文本框，通过它用户可以输入文本，下面是一个列表框，用来显示供用户进行选择的列表项。可以认为 TComboBox 就是文本框与列表框的组合，与文本框和列表框的功能基本一致。

与列表框相比，组合框不能够多选，因此它无 MultiSelect 属性和 ExtendedSelect 属性，但它有一个 Style 属性用来决定组合框的风格。下面仅介绍一下组合框组件的一些特殊属性。

- 1 DropDownCount 属性：用于设置组合框的列表框中所能显示的列表项的最大数目，默认情况下，列表框中可以显示 8 个列表项，当超过 8 个列表框时将显示出垂直滚动条。当希望在列表框中显示超过或少于 8 个的列表项时，就需要设置该属性。
- 1 Style 属性：用于设置或返回组合框的风格。当 Style 属性取值为 csDropDownList 时，组合框风格为下拉列表框；取值为 csDropDown 时，为下拉式组合框，即将文本框与下拉列表框组合在一起，用户可以输入新的数据，也可以单击向下的按钮来显示所有选项，然后从选项选择一个选项；取值为 csSimple 时，为简单组合框，即将文本框与列表框简单地组合在一起。

【例 7-4】 编写一个向列表框中添加选项的应用程序，程序的设计界面如图 7-8 所示，程序的运行界面如图 7-9 所示。程序运行时，在【输入】编辑框里输入任意文本并按 Enter 键后，如果该文本不在列表框就自动添加到列表框中，如果该文本已在列表框中就不添加到列表框并且将编辑框里的文本以高亮度显示，同时在列表框中选中的列表项。

【实现分析】

为判断在编辑框中按下的键是否为 Enter 键，可使用编辑框的 OnKeyPress 事件，通过该事件的参数 Key 来判断所按下的键是否为 Enter 键。为了判断编辑框的文本是否是列表框中的列表项，可以利用循环语句来实现。将编辑框的文本与列表框的列表项逐个比较，判断是否相等，如果相等说明该文本在列表框中，应终止整个循环，同时把该列表项选中。如果直到比较完列表框中的所有列表项，依旧无与编辑框中的字符串相等的选项，则应把编辑框中的字符串作为列表项添加到列表框中。要使编辑框中的文本高亮显示，只需要全部选中即可。



图 7-8 程序设计界面



图 7-9 程序运行界面

【界面设计】

窗体组件属性设置及其作用如表 7-5 所示。

表 7-5 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Caption	'输入'	提示输入选项
Label2	Caption	'添加'	提示添加选项
Edit1	Text	"	用来输入选项
ListBox1			添加选项
Form1	Caption	'查找列表项'	容纳其他组件

【程序代码】

```


procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
    mm: Integer;
begin
    if key=#13 then                                //Enter 键的 ASCII 值为 13
    begin
        mm:=0;
        while (mm<ListBox1.Count) do                //ListBox1.Count 为列表项的总数
        begin
            if Edit1.Text=ListBox1.Items[mm] then
            begin
                Edit1.SelectAll;                      //编辑框以高亮度显示
                ListBox1.ItemIndex :=mm;              //选中该列表项
                break;                                //终止整个循环
            end;
            mm:=mm+1;
        end;
        if mm=ListBox1.Count then                    //说明编辑框文本不在列表框中
        begin
            ListBox1.Items.Add(Edit1.Text);           //添加到列表框
            Edit1.Text:="";
        end;
    end;
end;

```



```
end;  
end;  
end;
```

7.1.4 TTimer 时钟组件的使用

TTimer 组件是个非可视组件，又称计时器组件或定时器组件，位于【System】选项卡中，图标是。该组件的主要作用是按一定时间间隔周期性地触发一个名为 OnTimer 的事件，因此在该事件的代码中可以放置一些需要每隔一段时间重复执行的程序段。

(1) TTimer 组件的属性

- **Enabled** 属性：用于设置程序运行时定时器是否正在运行。值为 **True** 时，计时器正在运行，值为 **False** 时，计时器不在运行。
- **Interval** 属性：用于设置定时器两次 OnTimer 事件发生的时间间隔，以毫秒为单位。如把它的值设置为 500，则将每隔 0.5 秒发生一个 OnTimer 事件。

(2) TTimer 组件的事件

- **OnTimer** 事件：在 Enabled 属性值为 **True** 时，该事件每隔一定时间间隔自动触发，触发的时间间隔由 Interval 属性指定。这也是 Timer 组件的惟一事件。

【例 7-5】 利用 TTimer 组件，显示系统的当前时间，要求每隔 0.5 秒显示一次。程序设计界面如图 7-10 所示，程序运行界面如图 7-11 所示。



图 7-10 程序设计界面



图 7-11 程序运行界面

【实现分析】

获取系统的当前时间可通过标准函数 Now 来实现，为显示系统时间，可使用一个 TTimer 组件，在该组件的 Timer 事件中调用 Now 函数获取系统的当前时间，然后把该时间转化为字符串显示出来即可，把时间类型转换成字符串类型可使用标准函数 TimeToStr 来实现。为使当前时间及时得到更新，TTimer 组件的 Interval 属性值应小于 500（即 0.5 秒）。

【界面设计】

给窗体添加一个 TTimer 组件和一个 TLabel 组件，设置 TTimer 组件的 Interval 属性值为 500。


【程序代码】

```
procedure TForm1.Timer1Timer(Sender: TObject);  
begin  
    Label1.Caption:=TimeToStr(now);           //将系统当前时间转化为字符串  
end;
```

7.1.5 对话框组件的使用

Dialogs 选项卡中提供了几个标准的对话框组件，下面介绍几个常用的对话框组件，包括 TOpenDialog（打开对话框）、TSaveDialog（保存对话框）、TFontDialog（字体对话框）和 TColorDialog（颜色对话框）。

1. TOpenDialog 组件

TOpenDialog 组件的图标是，用于显示打开对话框，供用户选择文件。当用户选择了某个文件，该文件的文件名（包括驱动器和路径）将被赋值给 TOpenDialog 组件的 FileName 属性。

（1）TOpenDialog 组件的常用属性

- **DefaultExt** 属性：用于指定默认的文件扩展名。如果选择了一个没有扩展名的文件或扩展名但没有在系统中注册的文件，系统将自动把该属性值作为文件的扩展名。
- **FileName** 属性：用于返回最近选中的文件的文件名，包括盘符和路径。
- **Files** 属性：用于返回选中的所有文件的文件名列表，Files 是一个 TStrings 型的属性，可以使用 TStrings 类型的相关方法。为了使打开对话框支持多选，应在它的 Option 集合属性中包含 ofAllowMultiSelect 成员。例如如下语句：

```
ListBox1.Items.Assign(OpenDialog1.Files);
```

该语句的作用是打开一个对话框并把选中的文件名列表显示在列表框中。

- **Filter** 属性：用于获取或设置当前文件名筛选器字符串，该字符串决定对话框的【另存为文件类型】或【文件类型】框中出现的选择内容。对于每个筛选选项，筛选器字符串都包含筛选器说明、垂直线条 (|) 和筛选器模式。不同筛选选项的字符串由垂直线条隔开。下面是筛选器字符串的一个示例：“文本文件 (*.txt)|*.txt|所有文件 (*.*)|*.*”。可以通过用分号来分隔各种文件类型，可以将多个筛选器模式添加到筛选器中。例如：“图像文件 (*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|所有文件 (*.*)|*.*”。
- **FilterIndex** 属性：用于获取或设置打开对话框中当前选定筛选器的索引。第一个筛选器的索引为 1，默认值为 1。
- **InitialDir** 属性：用于获取或设置文件对话框显示的初始目录。如果没有设置或设置的目录不存在，该属性值为当前工作目录。
- **Title** 属性：用于获取或设置对话框标题，如果没有给该属性赋值，则系统将使用默认标题：“打开”。

（2）TOpenDialog 组件的常用方法


TOpenDialog 组件的最常用方法只有一个 Execute。

- **Execute** 方法：用于显示打开对话框。若用户在打开对话框中单击了打开按钮时，该方法将返回 True，如果选择单击了 Cancel 或直接关闭对话框，该方法将返回 False。

打开对话框并没有实际打开文件，要想打开文件还必须通过代码来实现。

注意：在 Dialogs 选项卡中，还有一个 TOpenPictureDialog 组件，该组件的功能与使用方法基本与 TOpenDialog 一致，不同之处只是该对话框供用户选择图片文件。

2. TSaveDialog 组件


TSaveDialog 组件的图标是，主要用来弹出另存为对话框。TSaveDialog 具有前面所述的 TOpenDialog 组件的所有属性，且含义基本上是一样。需要注意的是：其 FileName 属性值为用户输入的要保存的文件名。

TSaveDialog 组件的常用方法是 Execute，该方法用来显示对话框。如果在该对话框中选择了文件并单击 Save 按钮，将返回一个值 True，如果单击 Cancel 按钮，将返回一个值 False。

保存对话框也没有实际保存文件，要想保存文件也必须通过代码来实现。


注意：在 Dialogs 选项卡中，还有一个 TSavePictureDialog 组件，该组件的功能与使用方法基本与 TSaveDialog 一致，不同之处只是该对话框供用户输入要保存的图片文件名。

3. TFontDialog 组件

TFontDialog 组件的图标是，用于弹出一个字体对话框，供用户对字体格式进行设置。它的一个最重要的属性是 Font，用来存放用户选择的字体格式。它的最常用方法是 Execute，调用该方法将弹出字体对话框，如果用户在弹出的字体对话框中选择了字体格式并单击 OK 按钮，该方法的返回值将为 True，如果单击 Cancel 按钮，该方法的返回值将为 False。

字体对话框也没有实际设定文本的字体，要设定文本的字体也必须通过代码来实现。

4. TColorDialog 组件

TColorDialog 组件的图标是，用于弹出一个颜色对话框，供用户选择颜色。它的一个最重要的属性是 Color 属性，用来存放用户选择的颜色。它的最常用的方法也是 Execute，调用该方法将弹出颜色对话框，如果在弹出的颜色对话框中选择了某种颜色并单击 OK 按钮，该方法的返回值将为 True，如果单击 Cancel 按钮，该方法的返回值将为 False。

颜色对话框也没有实际设定文本的颜色，要设定文本的颜色也必须通过代码来实现。

【例 7-6】 简单的文本编辑器。程序设计界面如图 7-12 所示。界面上有 4 个加速（SpeedButton）按钮，它们的功能依次是保存文件、打开文件、设置文本字体和设置文本颜色。界面上还有 4 个对话框组件 OpenDialog、SaveDialog、FontDialog 和 ColorDialog 组件与一个 Memo 组件。程序运行时，单击【保存】按钮，可以保存文件；单击【打开】按钮，可以打开一个文件；单击【字体】按钮，可以设置文本字体；单击【颜色】按钮，可以设置 Memo 组件中显示的文本颜色。程序运行界面如图 7-13 所示。



图 7-12 程序设计界面

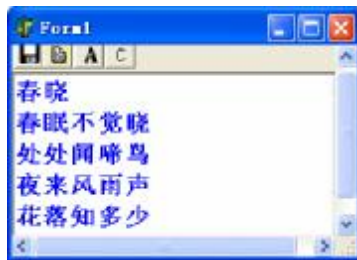


图 7-13 程序运行界面

【实现分析】

为了保存文件，可以通过 TSaveDialog 组件来弹出一个【另存为】对话框，供用户输入要保存的文件名，然后再利用 TMemo 组件的 Lines.SaveToFile 方法把 TMemo 组件中的内容

保存到用户输入的文件名中。为了打开一个已经存在的文件，可以通过 TOpenDialog 组件来弹出一个【打开】对话框，供用户选择一个文件，然后再利用 TMemo 组件的 Lines.LoadFromFile 方法将指定的文件装入到 TMemo 组件中。为改变 TMemo 组件中显示的文字的字体，可通过 TFontDialog 组件来弹出一个【字体】对话框，供用户选择字体，然后把【字体】对话框的 Font 属性赋值给 TMemo 组件的 Font 属性即可。同理，为改变 TMemo 组件中显示的文字颜色，可通过 TColorDialog 组件来弹出一个【颜色】对话框，供用户选择颜色，然后把【颜色】对话框的 Color 属性赋值给 TMemo 组件的 Color 属性即可。

【界面设计】

窗体组件属性设置及其作用如表 7-6 所示。

表 7-6 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
SpeedButton1	Name	SpeedButtonSave	单击可保存一个文件
SpeedButton2	Name	SpeedButtonOpen	单击可打开一个文件
SpeedButton3	Name	SpeedButtonFont	单击可改变备注框的字体
SpeedButton4	Name Caption	SpeedButtonColor 'C'	单击可改变备注框文本的颜色
Memo1	SrollBars	ssBoth	编辑文本
SaveDialog1			弹出【另存为】对话框
OpenDialog1			弹出【保存】对话框
FontDialog1			弹出【字体】对话框
ColorDialog1			弹出【颜色】对话框

【程序代码】

```
implementation
var
    fn:String;                                //定义单元级变量，用来存放文件名
    {$R *.dfm}
procedure TForm1.SpeedButtonSaveClick(Sender: TObject);
begin
    if SaveDialog1.Execute then
    begin
        fn:=SaveDialog1.FileName;
        Memo1.Lines.SaveToFile(fn);          //保存文件
        Form1.Caption:=fn;                   //窗体名为文件名（包含路径）
    end;
end;

procedure TForm1.SpeedButtonOpenClick(Sender: TObject);
begin
    if OpenDialog1.Execute then
    begin
        fn:=OpenDialog1.FileName;
```

```
Memo1.Lines.LoadFromFile(fn);           //将选择的文件读入到备注框中
Form1.Caption:=fn;
end;
end;
procedure TForm1.SpeedButtonFontClick(Sender: TObject);
begin
    if FontDialog1.Execute then
        Memo1.Font:=FontDialog1.Font;    //将字体赋值给备注框的字体
end;
procedure TForm1.SpeedButtonColorClick(Sender: TObject);
begin
    if ColorDialog1.Execute then
        Memo1.Font.Color:=ColorDialog1.Color;    //将颜色赋值给备注框的文本
end;
```

7.1.6 TImage 组件

TImage 组件用来显示图像，位于 Delphi 的 Additional 选项卡中，图标为.

1. TImage 组件的常用属性

- Picture 属性：用于设置在 Image 组件中显示的图片，方法是在对象浏览器中选中该属性，单击该属性后的【...】按钮，将会出现一个【Picture Editor】对话框，在该对话框中单击【Load】按钮，将会出现【Load Picture】对话框，在该对话框中选中一个图像文件后单击【打开】按钮，将会返回到【Picture Editor】对话框，在该对话框中单击【OK】按钮，对话框关闭，会发现图像已经显示在 TImage 组件中了。
- Stretch 属性：用于决定图片是否自动拉伸或缩小到与 TImage 组件一样大小，以填满 TImage 组件。取值为 True 时，表示图片将自动改变大小以填满 TImage 组件，取值为 False 时，图片原样显示，若 TImage 组件较小，图片将不能全部显示。如图 7-14 所示。



图 7-14 图像的显示

2. TImage 组件的常用方法

TImage 组件的方法一般不使用，而是使用它的 Picture 属性的方法。Picture 属性主要有两个方法。

- LoadFromFile 方法：用于把图片文件装载到 TImage 组件中，语法格式如下：

```
procedure LoadFromFile(const FileName: string);
```



其中，参数 `FileName` 为要装载的图片文件名。

■ **SaveToFile 方法**：用于把 `TImage` 组件中显示的图片保存到指定的文件中，语法格式如下：

```
procedure SaveToFile(const FileName: string);
```

其中，参数 `FileName` 为要保存图片的文件名。

7.1.7 菜单组件

Windows 的菜单系统是 GUI 的重要组成之一，在 Delphi 中有两类菜单：主菜单 `TMainMenu` 和弹出式菜单 `TPopupMenu`，它们都位于 `Standard` 选项卡中。主菜单的图标是，利用它可以创建类似于 Word、Delphi 等的主菜单。弹出式菜单的图标是，利用它可以创建类似于单击右键弹出的快捷菜单。下面以 `TMainMenu` 菜单为例讲解菜单的使用方法。

1. 菜单的结构

图 7-15 和图 7-16 所示为典型的菜单结构。其中有文字的单个命令称菜单项，顶层菜单项是横着排列的，单击某个菜单项后弹出的称为菜单或子菜单，它们均包含若干个菜单项，菜单项其实是 `TMenuItem` 类的一个对象。菜单项变灰显示时，表示该菜单项当前是被禁止使用的。有的菜单项的提示文字中有带下划线的字母，该字母称为热键（或访问键），若是顶层菜单可通过按“`Alt+热键`”打开该菜单，若是某个子菜单中的一个选项，则在打开子菜单后直接按热键就会执行相应的菜单命令。有的菜单项后面有一个按钮或组合键，称快捷键，在不打开菜单的情况下按快捷键，将执行相应的命令。在图 7-15 中，【保存文件】菜单项是加粗显示的，该菜单项称为默认项。图 7-15 的【另存为】和【退出】之间有一个灰色的线条，该线条称为分隔线或分隔符。在图 7-16 中菜单项【白色背景】前面有一个“√”号，称为选中标记，菜单项加上选中标记表示该菜单项代表的功能当前正在起作用。

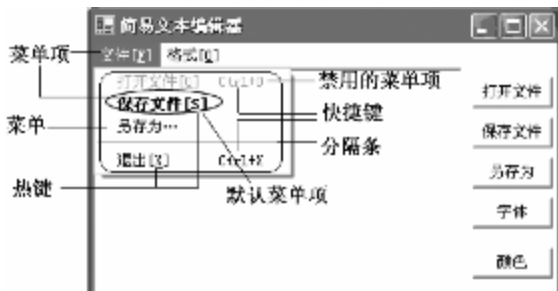


图 7-15 菜单结构一

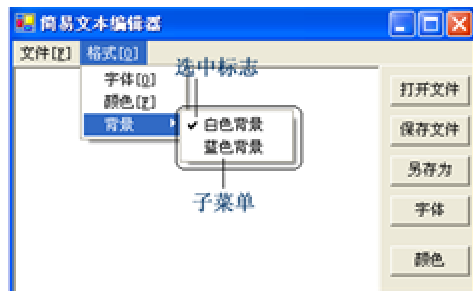



图 7-16 菜单结构二

2. 菜单项的常用属性

■ **Items 属性**：菜单项数组属性，用于存放菜单中的菜单项。设置菜单项的方法是：在窗体上添加一个 `TMainMenu` 组件，双击该组件或在对象观察器中单击 `Items` 属性，然后单击该属性右边的按钮，即可打开菜单编辑器，并自动生成一个 `TMenuItem` 对象，

如图 7-17 所示。蓝色的菜单项表示当前正在设置的菜单项，可以在对象观察器中设置该菜单项的属性。图 7-18 是一个设计好的菜单。此时单击【文件】右边的方框就可以设计菜单条中的下一菜单。

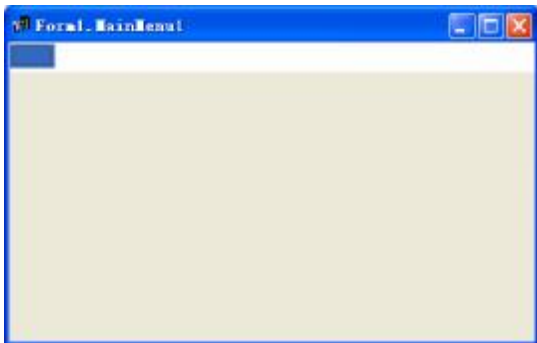


图 7-17 菜单编辑器

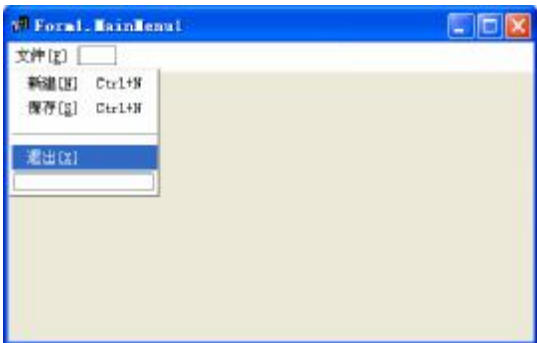


图 7-18 用菜单编辑器设计的菜单

- ! **Name** 属性：该属性代表的是菜单项名称。
- ! **Caption** 属性：用于设置或返回菜单项中显示的文字，如果要把属性中的某个字符设置为热键，可在该字符前添加一个符号“&”。如果把该属性值设置为“-”，该菜单项将是一个分隔条。
- ! **Visible** 属性：用于确定菜单项是否可见，值为 **True** 时，菜单项可见，值为 **False** 时，菜单项不可见。默认值为 **True**。
- ! **Checked** 属性：用于确定菜单项前是否显示选中标记，值为 **True** 时，表示该菜单项被选中，值为 **False** 时，表示该菜单项未被选中。默认值为 **False**。
- ! **ShortCut** 属性：用于确定菜单项的快捷键。
- ! **Enabled** 属性：用于确定菜单项是否可以使用，值为 **True** 时，表示可以使用，值为 **False** 时，菜单项将以变灰的形式显示，不可使用。
- ! **Items** 属性：如果该菜单项还有下一级子菜单，可通过该属性来访问子菜单中的菜单项，可看为一个 **TMenuItem** 型的数组，下标从 0 开始。
- ! **Count** 属性：如果该菜单项有下一级子菜单，可通过该属性指示下一级子菜单的菜单项的数目。

3. Items 属性的常用方法

- ! **Add** 方法：用于在 **Items** 的末尾再添加一个菜单项，语法格式如下：

```
procedure Add(Item: TMenuItem); overload;
```

其中，参数 **Item** 代表要添加的菜单项。

- ! **Delete** 方法：用于在 **Items** 中删除指定的菜单项，语法格式如下：

```
procedure Delete(Index: Integer);
```

其中参数 **Index** 代表要删除的菜单项的索引，从 0 开始。

- ! **Insert** 方法：用于在 **Items** 的指定位置插入一个菜单项，语法格式如下：


```
procedure Insert(Index: Integer; Item: TMenuItem);
```

其中，第一个参数 **Index** 代表要插入的位置，第二个参数代表要插入的菜单项。

❶ **Remove 方法**：用于删除指定的菜单项，这与 **Delete** 方法不同，语法格式如下：

```
procedure Remove(Item: TMenuItem);
```

其中，参数 **Item** 代表要删除的菜单项。该方法与 **Delete** 方法不同，**Delete** 方法的参数是菜单项的索引号，而 **Remove** 方法的参数是菜单项的名字。

❷ **Clear 属性**：用于删除 **Items** 属性中的所有菜单项。

【例 7-7】 编写一个菜单演示程序，程序的设计界面如图 7-19 所示。程序运行时，执行【File】→【Time】命令将显示系统当前时间；执行【File】→【Exit】命令或在窗体空白处单击鼠标右键，在弹出的菜单上选择【Exit】命令即可退出应用程序。程序运行界面如图 7-20 和图 7-21 所示。



图 7-19 程序设计界面

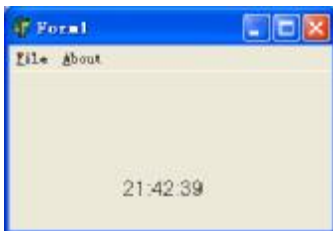


图 7-20 程序运行界面（一）



图 7-21 程序运行界面（二）

【实现分析】

在窗体上添加 4 个组件：**TMainMenu**、**TPopupMenu**、**TTimer** 和 **TLabel** 组件。双击 **TMainMenu**，添加两个菜单项 **File** 和 **About**，在 **File** 菜单项下再添加 2 个菜单：**Time** 和 **Exit**（菜单项名为 **Exit1**）。双击 **TPopupMenu** 组件，添加两个菜单：**Save** 和 **Exit**（菜单项名的 **Name** 属性值设置为 **Exit2**），并在对象观察器中设置窗体的 **PopupMenu** 属性为 **PopupMenu1**（这一步设置很重要）。单击【File】→【Time】菜单项，将生成有关事件代码框架，编写代码“**Label1.Caption:=TimeToStr(now);**”，这条语句用于显示系统的当前时间。单击【File】→【Exit】菜单项，编写代码“**Close;**”。在对象观察器中，选中【Exit2】菜单项，在其 **OnClick** 事件右边的空白处单击，选中 **Exit1Click**，共享该事件过程，从而在右键单击弹出菜单时单击【Exit】也可以使应用程序退出。

【程序代码】

```
procedure TForm1.Time1Click(Sender: TObject);
begin
    Label1.Caption:=TimeToStr(now);
end;
procedure TForm1.Exit1Click(Sender: TObject);
begin
    Close;
end;
```


例 7-7 是静态菜单设计,有时不一定能满足用户的要求,可能需要进行动态菜单设计,例 7-8 是一个动态菜单设计的例子。

【例 7-8】 动态菜单设计例。程序的设计界面如图 7-22 所示,程序运行时,单击【生成菜单】按钮,将动态生成一个【File】菜单,它包括 4 个菜单项: New、Open、Copy 和 Exit。程序运行界面如图 7-23 所示。



图 7-22 程序设计界面



图 7-23 程序运行界面

【实现分析】

首先创建一个主菜单,由于主菜单的基类型为 TMainMenu,因此可以调用 TMainMenu 的 Create 方法创建主菜单;其次添加 File 菜单,应调用 TMenuItem 的 Create 方法创建一个菜单项,再把它添加到 MainMenu 菜单中去;再次创建 4 个菜单项,由于菜单项的基类型为 TMenuItem,因此也可以调用 TMenuItem 的 Create 方法创建菜单项,创建过后把它们作为 File 菜单的子菜单添加进来。

【界面设计】



在窗体上添加一个 Button 组件,设置它的 Caption 属性值为“生成菜单”。

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  cd:TMainMenu;           //定义基类型为 TMainMenu 的变量
  ff:TMenuItem;           //定义基类型为 TMenuItem 的变量
  cdx:Array[1..4]of TMenuItem; //定义基类型为 TMenuItem 的数组
  k:Integer;
begin
  cd:=TMainMenu.Create(self); //创建主菜单
  ff:=TMenuItem.Create(self); //创建主菜单实例
  ff.Caption:='File';         //主菜单的名称为“File”
  cd.Items.Add(ff);           //添加主菜单从而显示出来
  for k:=1 to 4 do
    cdx[k]:=TMenuItem.Create(self); //给 File 菜单创建 4 个菜单项
    cdx[1].Caption:='New';          //给 File 菜单下的菜单项取名
    cdx[2].Caption:='Open';
    cdx[3].Caption:='Copy';
    cdx[4].Caption:='Exit';
  for k:=1 to 4 do
```

```
cd.Items[0].Add(cdx[k]);      //给 File 菜单添加 4 个菜单项  
end;
```

7.1.8 TTabControl 组件和 TPageControl 组件的使用

TTabControl 组件和 TPageControl 组件外观类似,都是选项卡组件,都位于 Win32 选项卡下。不过 TTabControl 是单选项卡组件,而 TPageControl 是多选项卡组件,它的每一个选项卡是一个独立的 TTabSheet 对象,因此可以在每个选项卡上放置不同的组件。TTabControl 组件的图标是, TPageControl 组件的图标是。两种组件的功能都是扩展屏幕的空间,使程序界面更简洁漂亮。

1. TPageControl 组件的使用

TPageControl 组件可以包含很多选项卡,每一个选项卡均可放置不同的组件以显示不同的内容,每一个选项卡均是一个 TTabSheet 对象。用户可以单击该组件顶上的标签来切换选项卡。刚刚创建的 TPageControl 组件不包含任何选项卡,如图 7-24 所示。要为 TPageControl 组件添加选项卡可在该组件上单击右键,在出现的快捷菜单中选中【New Page】菜单项,将为该组件添加一个选项卡。图 7-25 是添加了 3 个选项卡后的 TPageControl 组件。要选中某个选项卡,首先应单击该选项卡的标签,然后再在选项卡中部单击。选中了某选项卡后,还可以在选项卡上用鼠标右键单击,在弹出的菜单上选择【Delete Page】删除选中的选项卡,或直接按 Delete 键也可以删除选中的选项卡。

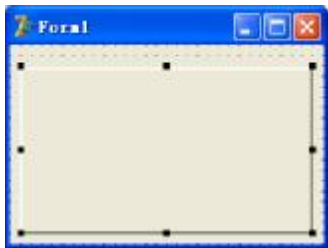


图 7-24 没有添加选项卡的 PageControl 组件



图 7-25 添加 3 个选项卡后的 PageControl 组件

一般很少使用该组件的方法,下面主要介绍一下该组件的常用属性和事件。

(1) TPageControl 组件的常用属性

- **Pages** 属性: 该属性是一个数组属性,它的每一个元素代表 TPageControl 组件中的一个选项卡,是 TTabSheet 类型的。访问某选项卡的一般形式是: Pages[Index], 其中参数 Index 代表选项卡的索引号,从 0 开始。
- **PageCount** 属性: 用于返回 TPageControl 组件中选项卡的数量。
- **ActivePage** 属性: 用于为 TPageControl 组件设置当前选项卡或返回 TPageControl 组件的当前选项卡,其类型是 TTabSheet。
- **ActivePageIndex** 属性: 用于返回 TPageControl 组件的当前选项卡的索引号,或通过该属性把某索引号的选项卡设为当前选项卡。
- **MultiLine** 属性: 用于决定标签是否允许多行显示,值为 True 时,允许多行显示,值为 False 时,不允许多行显示。在该属性值为 False 时,在选项卡数量较多的情况下,有

些选项卡的标签将无法显示出来。

(2) TPageControl 组件的常用事件

- ! OnChange 事件：该事件在选择了一个选项卡后发生。
- ! OnPageChanging 事件：该事件在某选项卡被选中前发生。
- ! OnChanging 事件：在另一个选项卡被选中之前发生。

【例 7-9】 编写一个显示文本文件和图片文件的应用程序,要求使用 TPageControl 组件,在 TPageControl 组件的两个选项卡中分别显示文本文件和图片文件的内容。程序的设计界面如图 7-26 和图 7-27 所示。程序的运行时,单击【文本文件】选项卡,然后单击【打开】按钮将弹出【打开】对话框供用户选择一个文本文件,选中的文本文件将显示在 TMemo 组件中,如图 7-28 所示。单击【图片文件】选项卡,然后单击【打开】按钮将弹出【打开图片】对话框供用户选择一个图片文件,选中的图片文件将显示在 TImage 组件中,如图 7-29 所示。



图 7-26 程序设计界面 (一)



图 7-27 程序设计界面 (二)



图 7-28 程序运行界面 (一)



图 7-29 程序运行界面 (二)

【实现分析】

要使用 TPageControl 组件,首先应给它添加选项卡,添加方法是在该组件上单击右键,在出现的快捷菜单中选择【New Page】命令,本例添加了两个选项卡。两个选项卡的名称分别为 TabSheet1 和 TabSheet2,分别选中这两个选项卡,设置它们的 Caption 属性值分别为“文本文件”和“图片文件”。要向某一选项卡添加组件,首先应选中该选项卡,然后再添加组件。本例在【TabSheet1】选项卡中添加了一个 TMemo 组件和一个 TButton 组件(组件名为 Button1),在【TabSheet2】选项卡中添加了一个 TImage 组件和一个 TButton 组件(组件名为 Button2)。为显示【打开】对话框和【打开图片】对话框,可通过给应用程序添加 TOpenDialog 和 TOpenPictureDialog 组件来实现。

【界面设计】

窗体组件属性设置及其作用如表 7-7 所示。

表 7-7 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
PageControl1			页框组件，提供两个选项卡分别显示图片文件和文本文件
TabSheet1	Caption	'文本文件'	该选项卡用来打开并显示文本文件
Memo1	Lines	"	用来显示打开的文本文件
Button1	Caption	'打开'	单击它将弹出一个【打开】对话框供用户选择文本文件
TabSheet2	Caption	'图片文件'	单击它将弹出一个【打开文件】对话框供用户选择图片文件
OpenDialog1			显示【打开】对话框
OpenPictureDialog1			显示【打开图片】对话框

【程序代码】

```
var
  Form1: TForm1;
  TxtFName, PicFName: String;           //存放打开的文本文件名和图片文件名的变量
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
  OpenDialog1.Filter := '文本文件(*.txt)|*.txt';
  if(OpenDialog1.Execute) then
  begin
    TxtFName := OpenDialog1.FileName;    //弹出打开文件对话框，供用户选择文件
    Memo1.Lines.LoadFromFile(TxtFName);  //把选中的文件内容装入 Memo 组件中
  end;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  OpenPictureDialog1.Filter := '图片文件(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF';
  if(OpenPictureDialog1.Execute) then
  begin
    PicFName := OpenPictureDialog1.FileName;
                                           //弹出打开图片对话框，供用户选择图片文件
    Image1.Picture.LoadFromFile (PicFName);
                                           //把选中的图片文件显示在 Image 组件中
  end;
end;
```

2. TTabControl 组件的使用

与 TPageControl 组件不同，TTabControl 组件并不是由很多选项卡组成，只是一个单一的对象，该对象可以有很多标签，从而形成“多选项卡”效果。添加到 TTabControl 组件中的组件将在选择每个标签时均会出现，但可以让这些组件显示不同的内容，从而创建类似于“多

选项卡”的效果。

创建一个新的 TTabControl 组件时只有一个选项卡，要为该组件上添加标签，只能通过 Tabs 属性来进行。创建方法是：在对象观察器中选中 Tabs 属性，然后单击其后的【…】按钮，将会打开【String List Editor】对话框，在该对话框中输入每个标签的标题即可。

(1) TTabControl 组件的常用属性

- ▮ Tabs 属性：一个集合属性，它的每个元素对应一个 TTabControl 组件中的一个 TTab 对象。
- ▮ TabIndex 属性：用于设置或返回选中的 TTab 对象的索引，第一个 TTab 对象的索引为 0。如果没有 TTab 对象被选中，该属性值为-1。
- ▮ MultiLine 属性：作用同 TPageControl 组件的同名属性。

(2) TTabControl 组件的常用事件

- ▮ OnChanging 事件：用户单击某个选项卡，在切换到该选项卡之前将触发该事件。
- ▮ OnChange 事件：从一个选项卡切换到另一个选项卡之后将触发该事件。

【例 7-10】 编写一个日期与时间切换程序。程序的设计界面如图 7-30 所示，程序运行时，单击【日期】选项卡，将显示出当前日期，如图 7-31 所示，单击【时间】选项卡，将显示出当前时间，如图 7-32 所示。



图 7-30 程序设计界面



图 7-31 程序运行界面（一）



图 7-32 程序运行界面（二）

【实现分析】

由于日期和时间均可以在一个 TLabel 组件中显示，所以本例可使用 TTabControl 组件，通过它的 TabIndex 属性值来决定在 TLabel 组件中是显示时间还是显示日期。

【界面设计】

窗体组件属性设置及其作用如表 7-8 所示。

表 7-8 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
TabControl1	Tabs	'日期' '时间'	用来产生“两个选项卡”的效果
Label1			通过其 Caption 属性来显示日期或时间
Timer1	Interval	500	每隔一定的时间间隔（0.5 秒）刷新时间和日期值

【程序代码】

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if TabControl1.TabIndex=0 then
```


```

Label1.Caption :=FormatDateTime('yyyy-mm-dd',now)           //显示日期
else
    Label1.Caption :=FormatDateTime('hh:mm:ss',now);         //显示时间
end;

```

7.1.9 TScrollBar、TTrackBar 和 TProgressBar 组件的使用

1. TScrollBar 组件的使用

TScrollBar 组件又称滚动条组件，位于【Standard】选项卡中，图标是。TScrollBar 组件的作用是控制窗口和组件内容的滚动显示。一般很少使用该组件的方法，下面着重介绍一下该组件的属性和事件。

(1) TScrollBar 组件的常用属性

- ┌ Max 属性：用于设定滚动条组件的最大值。
- ┌ Min 属性：用于设定滚动条组件的最小值。
- ┌ Position 属性：用于设置或获取滚动块当前位置的值。
- ┌ LargeChange 属性：用于设置单击滚动块两边的位置时，Position 属性增加或减少的值。
- ┌ SmallChange 属性：用于设置单击滚动块两端的箭头时，Position 属性增加或减少的值。

(2) 滚动条组件的常用事件

- ┌ OnChange 事件：当滚动条的 Position 属性值发生变化时将触发该事件。
- ┌ OnScroll 事件：当用鼠标或键盘滚动了滚动条时触发该事件。

【例 7-11】 编写一个利用滚动条输入数字的应用程序。程序的设计界面如图 7-33 所示。程序运行时当改变滚动条滑块位置时，将把滚动条当前位置的值显示在编辑框中。当在编辑框中输入数字时，滚动条的滑块将移动到相应的位置。程序的运行界面如图 7-34 所示。



图 7-33 程序设计界面



图 7-34 程序运行界面

【实现分析】

当 TScrollBar 组件的滑块位置发生变化时将会发生 OnChange 事件，可在该事件中把滚动条组件的 Position 属性值转换为 String 类型并赋值给 TEdit 组件的 Text 属性，从而实现用滚动条向编辑框中输入数值的功能。当 TEdit 组件中输入的内容发生变化时将会发生 OnChange 事件，可在该事件中把 TEdit 组件的 Text 属性转换为 Integer 类型并赋值给 TScrollBar 组件的 Position 属性，从而实现利用编辑框组件设置滚动条滑块位置的功能。

【界面设计】

窗体组件属性设置及其作用如表 7-9 所示。

表 7-9 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Text	'值'	提示
Label2	Text	'0'	提示能够输入的最小值
Label3	Text	'1000'	提示能够输入的最大值
ScrollBar1	Min	0	通过滚动条输入数字
	Max	1000	
	LargeChange	50	
	SmallChange	10	
Edit1	Text	"	输入或显示数字


【程序代码】

```

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    Edit1.Text :=inttostr(ScrollBar1.Position );           //在编辑框中显示滚动条的位置值
end;
procedure TForm1.Edit1Change(Sender: TObject);
begin
    if edit1.text="" then
        scrollBar1.Position :=0                           //如果编辑框的文本为空，设滚动条的位置值
为 0
    else
        ScrollBar1.Position :=strtoint(Edit1.text);       //用滚动条的位置值反映输入的数字
end;

```

2. TProgressBar 组件的使用

TProgressBar 组件又称进度条，位于 Wn32 选项卡中，图标为 。该组件可在水平栏中显示适当长度的矩形来指示进程的进度。当执行进程时，进度条用系统突出显示颜色的矩形框在水平栏中从左向右进行填充。进程完成时，进度栏被填满。当某进程运行时间较长时，如果没有视觉提示，用户可能会认为应用程序不响应，通过在应用程序中使用进度条，就可以告诉用户应用程序正在执行冗长的任务且应用程序仍在响应。

(1) TProgressBar 组件的常用属性

- Max 属性：用于定义 TProgressBar 组件的上限，当进度栏被填满时 Position 属性的值等于该属性的值。
- Min 属性：用于定义 TProgressBar 组件的下限，在进度栏开始填充时，Position 属性的值等于该属性的值。
- Orientation 属性：用于定义 TProgressBar 组件是垂直排列的还是水平排列的。取值为 pbHorizontal，表示是水平排列的，取值为 pbVertical，表示是垂直排列的。
- Position 属性：代表 TProgressBar 组件的当前位置值，当该值等于 Min 时，TProgressBar 组件未被填充，当该值等于 Max 时，TProgressBar 组件被填满。
- Step 属性：用于设置当调用 TProgressBar 组件的 StepIt 方法时，Position 属性的增加值。

(2) TProgressBar 组件的常用方法

┆ StepBy 方法：使 TProgressBar 组件的 Position 属性值增加一定的数量。其语法如下：


```
procedure StepBy(Delta: Integer);
```

其中，参数 Delta 表示要增加的数值。

┆ StepIt 方法：使 TProgressBar 组件的 Position 属性值增加一个固定的值，该值由 Step 属性指定，该方法无参数。

第13章中的有关多媒体的例子中，有关于 TProgressBar 组件应用的典型例子，此处不再赘述。

3. TTrackBar 组件的使用

TTrackBar 组件位于【Win32】选项卡中，它的图标是。该组件包含一个滑块和一个刻度标记，通常用于通过拖动滑块，来改变某个数值。一般不使用该方法。

(1) TTrackBar 组件的常用属性

┆ Max 属性：用于设置 TTrackBar 组件 Position 属性的最大值。

┆ Min 属性：用于设置 TTrackBar 组件 Position 属性的最小值。

┆ Position 属性：代表滑块所在位置的值。

┆ SelStart 属性：用于设置滑块拖动范围的起始点。

┆ SelEnd 属性：用于设置滑块拖动范围的终止点。

┆ Frequency 属性：用于设置刻度标记的频率，此频率与取值范围有关。

┆ LineSize 属性：用于设置当按箭头键时，TTrackBar 组件的 Position 属性增加或减少的值。按向下箭头或左箭头时，Position 属性值将按该属性的值减少，按向上箭头或右箭头时，Position 属性值将按该属性的值增加。



┆ PageSize 属性：用于设置当按 PageDown、PageUp 键时或在 TTrackBar 组件上单击时，TTrackBar 组件的 Position 属性增加或减少的值。按 PageDown 键时，Position 属性值减少一个 PageSize 属性的值，按 PageUp 键时，Position 属性值将增加一个 PageSize 属性的值。

┆ Orientation 属性：用于定义 TTrackBar 组件是水平排列的还是垂直排列的。取值为 trHorizontal 表示是水平排列的，取值为 trVertical 表示是垂直排列的。

(2) TTrackBar 组件的常用事件

TTrackBar 组件的常用事件是 OnChange，该事件在 Position 属性值发生改变时触发。

7.1.10 TPanel 组件和 TGroupBox 组件

TPanel 组件位于【Standard】选项卡下，图标是。TGroupBox 组件位于【Standard】选项卡中，图标是。这两个组件都是容器组件（又称父组件），可以包含其他组件（又称子组件），并能对子组件发生影响。要想让父组件真正包容子组件，必须首先绘制父组件并选中，再在父组件中添加子组件。一般不使用这两种组件的方法和事件。


TPanel 组件的常用属性如下。


┆ BevelInner 属性：用于设置面板内层边框斜面的类型。

- ! BevelOuter 属性：用于设置外层边框斜面的类型。
- ! BevelWidth 属性：用于设置斜面的宽度。
- ! BorderStyle 属性：用于设置边框类型。

TGroupBox 组件常用来作为一组单选按钮或复选框的容器，一般只使用它的 Caption 属性。

7.1.11 工具栏组件与状态栏组件

关于工具栏，这里只是简单介绍。用户可以利用 TPanel 组件和 TSpeedButton 组件生成工具栏，也可以利用 Delphi 提供的 TToolBar 组件（位于【Win32】选项卡下，图标是）生成工具栏。在窗体上添加 TToolBar 组件，在其上用鼠标右键单击，在弹出的菜单中选择【New Button】可以生成一个新按钮，选择【New Separator】可以生成一个新的分隔符。不同分隔符之间的按钮属于不同的组，当同一组按钮的 Group 属性设置为 True 时可以使它们成组方式工作。当设置按钮的 ShowCaption 为 True 时可以使按钮显示标题（默认时不显示标题）。

状态栏组件 TStatusBar 也位于 Win32 选项卡下，图标是，它的主要作用是说明当前应用程序的一些状态。为了给状态栏添加子面板或删除状态栏中的子面板，可以用鼠标右键单击 TStatusBar 组件，在弹出的菜单中单击【Panels Editor】菜单项，或者在对象观察器中双击 Panels，两种方法均会打开【Edit StatusBar1.Panels】对话框，在该对话框中单击右键，通过快捷菜单就可以添加或删除子面板了，如图 7-35 所示。

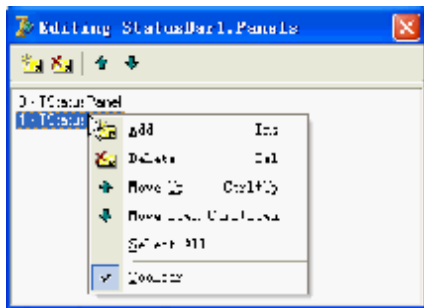


图 7-35 【Edit StatusBar1.Panels】对话框

7.2 典型实例

7.2.1 典型实例一

【实例题目】：文本格式设置程序

编写一个设置编辑框组件中显示的文本格式的应用程序，程序的设计界面如图 7-36 所示。程序运行时，如果选中了【编辑框 1】复选框，字体设置将是针对第一个编辑框进行的，如果选中了【编辑框 2】复选框，字体设置将是针对第二个编辑框进行的，如果两个复选框均选中，

字体设置将是针对两个编辑框进行的。下部的三个单选按钮组分别用来设置相应编辑框中显示文本的字体、大小和颜色，注意：只有在单击【确定】按钮后，字体格式设置才生效。图 7-37 是某一时刻的运行界面。



图 7-36 程序设计界面



图 7-37 程序运行界面

【实现方法】

可在【确定】按钮的 OnClick 事件过程中，首先测试 CheckBox1 是否被选中（通过它的 Checked 属性值），如果选中，再判断三个单选按钮组中各选中了哪个单选按钮（通过它的 ItemIndex 属性值），并给第一个编辑框的 Font 属性的相应子属性（Name、Size 和 Color）赋相应的值。同理测试 CheckBox2 是否被选中，如果选中，则再判断三个单选按钮组中各选中了哪个单选按钮，并设置第二个编辑框的 Font 属性的相应子属性值。

【界面设计】

窗体组件属性设置及其作用如表 7-10 所示。

表 7-10 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Edit1	Text	'直面…态度'，见图 7-36	显示文本及格式
Edit2	Text	'从挫…好汉'，见图 7-36	显示文本及格式
CheckBox1	Caption	'编辑框 1'	选中它给编辑框 1 设置格式
CheckBox2	Caption	'编辑框 2'	选中它给编辑框 2 设置格式
RadioGroup1	Caption Items	'字体' '宋体' '幼圆' '黑体'	设置编辑框中显示的文本的字体名
RadioGroup2	Caption Items	'大小' '12' '16' '18'	设置编辑框中显示的文本的字体大小
RadioGroup3	Caption Items	'颜色' '红色' '蓝色' '绿色'	设置编辑框中显示的文本的颜色

【程序代码】

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    CheckBox1.Checked := True;           //选中复选框 1
    RadioGroup1.ItemIndex := 0;          //选中宋体
    RadioGroup2.ItemIndex := 0;          //选中 12
    RadioGroup3.ItemIndex := 0;          //选中红色
    Edit1.Font.Name := '宋体';
    Edit1.Font.Size := 12;
    Edit1.Font.Color := clRed;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    if CheckBox1.Checked = true then
        //如果 CheckBox1 选中,则设置 Edit1 中显示文本的格式
        begin
            Edit1.Font.Name := RadioGroup1.Items[RadioGroup1.ItemIndex]; //设置字体
            Edit1.Font.Size := strToInt(RadioGroup2.Items[RadioGroup2.ItemIndex]); //设置字的大小
            case RadioGroup3.ItemIndex of //设置颜色
                0: Edit1.Font.Color := clRed;
                1: Edit1.Font.Color := clBlue;
                2: Edit1.Font.Color := clBlue;
            end;
        end;
    if CheckBox2.Checked = true then
        //如果 CheckBox2 选中,则设置 Edit2 中显示文本的格式
        begin
            Edit2.Font.Name := RadioGroup1.Items[RadioGroup1.ItemIndex]; //设置字体
            Edit2.Font.Size := strToInt(RadioGroup2.Items[RadioGroup2.ItemIndex]); //设置字的大小
            case RadioGroup3.ItemIndex of //设置颜色
                0: Edit2.Font.Color := clRed;
                1: Edit2.Font.Color := clBlue;
                2: Edit2.Font.Color := clBlue;
            end;
        end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Application.Terminate ;             //退出
end;
```

7.2.2 典型实例二

【实例题目】：课程选择程序

编写一个课程选择程序，程序的设计界面如图 7-38 所示，左边的列表框显示可以选修的课程，右边的列表框中显示用户准备选修的课程。程序运行时，选中左边列表框中的一个或多个列表项，单击【>】按钮，选中的列表项将被移动到右边列表框中，同时选中的项从左边列表框中消失；同样，选中右边列表框里的一个或多个列表项，单击【<】按钮，选中的列表项将被移动到左边列表框中，同时选中的项从右边列表框中消失。单击【>>】按钮将把左边列表框中的所有列表项全部移到右边的列表框中，同时左边的列表框清空；单击【<<】按钮将把右边列表框中的所有列表项全部移到左边的列表框中，同时右边的列表框清空。程序某时刻的运行界面如图 7-39 所示。



图 7-38 程序设计界面



图 7-39 程序运行界面

【实现方法】

对于列表框组件，为实现多选，可把它的 `MultiSelect` 属性设置为 `True`（要实现扩展选择，还应把它的 `ExtendedSelect` 属性值设置为 `True`）。列表框组件有一个 `Selected` 属性，该属性是一个逻辑数组属性，用来指出相应位置的选项是否被选中。因此，为把一个列表框中选中的选项移动到另一个列表框中，可通过一个 `For` 循环，循环变量（设为 `i`）的值从项数（可通过它的 `Count` 属性值得到）减 1 直到 0 为止，判断以 `i` 为序号的那一项是否被选中，如果选中将它添加到另一列表框中（`Items.Add` 方法），同时把该选项从该列表框中删除（`Items.Delete` 方法）。要从一个列表框中把所有的列表项移动到另一个列表框中，可通过一个循环依次把列表框中的列表项添加到另一个列表框，然后清除本列表框中的列表项即可。

【界面设计】

窗体组件属性设置及其作用如表 7-11 所示。

表 7-11 组件属性及组件功能

组 件	属 性 名	属 性 值	作 用
Label1	Text	'可选课程: '	提示文字
ListBox1	Items	见图 7-39	显示可选课程供用户选择
	MultiSelect	True	
	ExtendedSelect	True	
Label2	Text	'已选课程: '	提示文字

续表

组 件	属 性 名	属 性 值	作 用
ListBox2	Items	"	显示用户已选课程
	MultiSelect	True	
	ExtendedSelect	True	
Button1	Name	ButtoRS	把左边列表框中选中的列表项移动到右边的列表框中
	Caption	'>'	
Button2	Name	ButtoRSA	把左边列表框中的全部列表项移动到右边的列表框中
	Caption	'>>'	
Button3	Name	ButtoLS	把右边列表框中选中的列表项移动到左边的列表框中
	Caption	'<'	
Button4	Name	ButtoLSA	把右边列表框中的全部列表项移动到左边的列表框中
	Caption	'<<'	

【程序代码】

```

procedure TForm1.ButtonRSClick(Sender: TObject);
var
  i:integer;
begin
  For i:=ListBox1.Count-1  downto 0 do
    //遍历 ListBox1 的所有列表项，i 代表列表项的序号
    begin
      if listBox1.Selected[i]  then
        //如果 i 项被选中
        begin
          listBox2.Items.Add(listBox1.Items.Strings[i]); //把该项添加到 ListBox2 中
          ListBox1.Items.Delete(i); //从该列表框中删除该项
        end;
      end;
    ButtonLS.Enabled :=True; //左移按钮可用
    ButtonLSA.Enabled :=True; //全部左移按钮可用
    if ListBox1.Count =0 then
      //如果 ListBox1 中已经没有列表项
      begin
        ButtonRS.Enabled :=False; //右移按钮不可以用
        ButtonRSA.Enabled :=False; //全部右移按钮不可用
      end;
    end;
  end;
procedure TForm1.ButtonRSAClick(Sender: TObject);
var
  i:integer;
begin
  For i:=0  to  ListBox1.Count-1 do
    //遍历 ListBox1 的所有列表项，i 代表列表项的序号
    listBox2.Items.Add(listBox1.Items.Strings[i]); //把该项添加到 ListBox2 中
    ListBox1.Clear ; //清除所有列表项
    ButtonLS.Enabled :=True; //左移按钮可用

```

```

ButtonLSA.Enabled :=True;           //全部左移按钮可用
ButtonRS.Enabled :=False;          //右移按钮不可以用
ButtonRSA.Enabled :=False;         //全部右移按钮不可用
end;

```

ButtonLS 和 ButtonLSA 的 OnClick 事件代码与上述代码类似，请自己完成。

7.3 上机练习

7.3.1 上机练习一

【练习题目】：文本的自动移动与手动移动

编写一个文本移动程序，程序的设计界面如图 7-40 所示。程序运行时，单击【自动移动】按钮，文本“欢迎各位专家莅临指导！”从右到左自动移动，移出窗体左边界时又从右边移入，如此循环。单击【手动移动】按钮，文本停止移动，但在按下左箭头键和 Alt 键时又将从右向左移动，移出左边界时也从右边移入。程序运行界面如图 7-41 所示。



图 7-40 程序设计界面

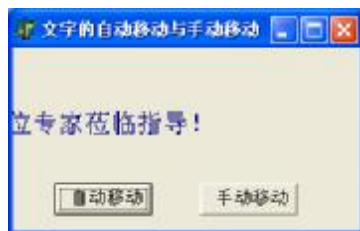


图 7-41 程序运行界面

【要点提示】

为了让文本自动地从右向左移动，可以添加一个 TTimer 组件，每隔一定时间触发一次 OnTimer 事件，在该事件中让标签组件的 Left 属性减少一定的数值。为了让标签组件在移出窗体的左边界时又移入窗体，可通过让标签组件的 Left 属性值等于窗体的宽度来实现。

为了能够手动控制文本的移动，可通过窗体的 OnKeyDown 来实现。该事件有一个参数 Key，通过它可以测试所按下的键，当参数 Key 为虚拟键 vk_Left 就表示按下左箭头键。为了在按下左箭头键和 Alt 键时移动文本，处理的方法与自动移动文本相同，也是让 TLabel 组件的 Left 属性值在每按一下左箭头键时减少一定数值。但移动文本的条件将变成：(Key= vk_Left)And(ssAlt in Shift)And(Timer1.Enabled=False)，即按下左箭头键和 Alt 键，同时时钟组件不可用的情况下，文本向左移动。

【参考代码】

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Left:=Form1.Width ;           //标签组件设置在右边界之外

```

```

Timer1.Enabled:=True;           //启动时钟组件，以自动移动文本
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    if (Label1.Left+Label1.Width>0) then //如果没有移出窗体
        Label1.Left:=Label1.Left-20    //Left 属性值减 20
    else
        Label1.Left:=Form1.Width;       //移出左边界时从窗体右边界移入
end;
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    if (Key=vk_Left)And(ssAlt in Shift)And(Timer1.Enabled=False) then
        //表示按下左箭头键和 Alt 键，同时时钟组件不可用

        if Label1.Left+Label1.Width>0 then
            Label1.Left:=Label1.Left-2
        else
            Label1.Left:=Form1.Width;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    Label1.Left:=Form1.Width;           //标签组件设置在右边界之外
    Timer1.Enabled:=False;
end;

```

7.3.2 上机练习二

【练习题目】：计算机硬件配置程序

设计一个计算机硬件配置程序，程序的设计界面如图 7-42 所示。程序运行时选中相应的配件后单击【确定】按钮，将把配置情况显示在左侧的列表框中，如图 7-43 所示。



图 7-42 程序设计界面



图 7-43 程序运行界面

【要点提示】

由于电脑品牌有很多，故可用一个组合框列出一些常用的品牌供用户选择，且允许用户

输入新的品牌。CPU 型号的选择及内存容量的选择是互斥的, 因此可用两个单选按钮组来实现。各种其他设备和配件的选择, 可以选也可以不选, 不互斥, 因此可用多个复选框来实现。在【确定】按钮的 Click 事件中, 可通过判断是否选择了电脑品牌 (Text 属性值是否为空), 如果选择了则把它加载到右侧的列表框中。判断在单选按钮组中选择了哪个单选按钮, 可通过测试它的 ItemIndex 属性值来实现。判断复选框是否被选中, 可通过测试它的 Checked 属性值来实现。

【参考代码】

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ListBox1.Clear ;
    if ComboBox1.Text <>'' then
        //如果选择或输入了电脑品牌, 则把电脑品牌加到列表框中
        ListBox1.Items.add(ComboBox1.Text );
    if RadioGroup1.ItemIndex <>-1 then
        //如果选择了 CPU 型号, 把型号值加载到列表框中
        ListBox1.Items.Add(RadioGroup1.items[RadioGroup1.ItemIndex]);
    if RadioGroup2.ItemIndex <>-1 then
        //如果选择了内存容量, 把内存容量值加载到列表框中
        ListBox1.Items.Add(RadioGroup2.items[RadioGroup2.ItemIndex]);
    if CheckBox1.Checked then
        //以下测试是否选择了某个具体设备或配件, 选择了就把它加到列表框中
        ListBox1.Items.Add('光驱');
    if CheckBox2.Checked then
        ListBox1.Items.Add('声卡');
    if CheckBox3.Checked then
        ListBox1.Items.Add('Modem');
    if CheckBox4.Checked then
        ListBox1.Items.Add('音箱');
end;
```

课后考场

一、选择题 (20 分, 每题 5 分)

1. 文本组件中, TEdit 组件和 TMemo 组件的一个很大区别是_____。
A. 前者能输入数据, 后者不能 B. 前者处理单行数据, 后者能处理多行数据
C. 前者不能编辑, 后者可以 D. 前者处理多行数据, 后者不能
2. 在设计时, 向 TRadioGroup 组件添加单选按钮只能通过编辑_____属性。
A. Items B. Name C. Caption D. ShowHint
3. 对于 TOpenDialog 组件, 当选择一个文件并打开时, 该文件所在的驱动器名、路径名、文件名和文件扩展名将被赋值给 TOpenDialog 组件的_____属性。

- A. Files B. Create C. FileName D. Destroy

4. 在窗体上添加弹出菜单组件 TPopupMenu 后, 为了使该组件有用, 除了使用菜单设计器添加菜单项, 编写正确的代码外, 还必须设置窗体的_____属性。

- A. Menu B. Name C. Caption D. PopupMenu

二、填空题 (40 分, 每空 5 分)

1. TImage 组件有一个_____属性, 当该属性值为 True 时, 加载的图片将自动改变大小以填满整个图片框。

2. 将滚动条 ScrollBar1 的当前值在编辑框 Edit1 中显示出来, 使用的语句是_____。

3. 下面语句的功能是_____。

```
Button1.ShowHint:=True;
```

```
Button1.Hint:='Hello';
```

4. 把 Label1 中显示文字的字体设置为隶书, 使用的语句是_____。

5. TTimer 组件有一个事件, 该事件每隔一定的时间间隔周期性地发生, 该事件是_____。

6. TTabControl 和 TPageControl 组件中有一个是单页面组件, 该组件是_____。

7. 分析下列程序段:

```
var
  k:Integer;
  t:TMainMenu;
  tt:Array[0..2]of TMenuItem;
begin
  t:=TMainMenu.Create(self);
  for k:=0 to 2 do
  begin
    tt[k]:=TMenuItem.Create(self);
    tt[k].Caption:='t'+IntToStr(k);
    t.Items.Add(tt[k]);
  end;
end;
```

其中, 语句 t:=TMainMenu.Create(self);的作用是_____, 语句 t.Items.Add(tt[k]);的作用是_____。

三、程序设计题 (40 分, 每题 20 分)

1. 编写一个文本编辑器程序, 程序的设计界面如图 7-44 所示, 其中 Format 菜单中仅有一个菜单项 Font。程序运行时, 单击【File】→【New】菜单项可以新建一个文档; 单击【File】→【Open】菜单项可以打开一个文档; 单击【File】→【Save】菜单项可以保存一个文档; 单击【File】→【Exit】菜单项可以退出程序; 单击【Format】→【Font】菜单项可以改变文档字体的显示格式。程序运行界面如图 7-45 所示。



图 7-44 程序设计界面

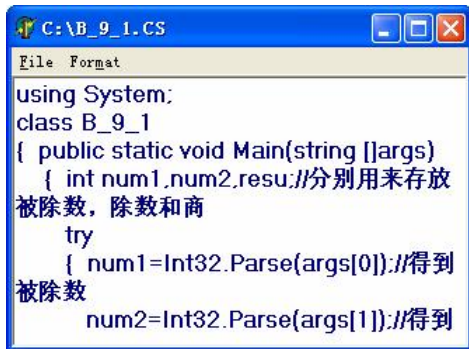


图 7-45 程序运行界面

2. 任意给定一个银行利率和本金, 计算到期利息与本金总和, 并用滚动条显示出来。如果拖动本金和利率滚动条, 本金和利率框里的数字也会随之而变。程序设计界面如图 7-46 所示, 程序运行界面如图 7-47 所示。



图 7-46 程序设计界面



图 7-47 程序运行界面

第 8 章 Delphi 7 的文件系统

本章要点

- 文件的概念与文件类型
- 在 Delphi 7 中打开、关闭和读写文本文件的方法
- 在 Delphi 7 中打开、关闭和读写记录文件的方法
- 与文件有关的函数的使用

8.1 理论知识

8.1.1 文件的基本概念

通常所说的文件是指存放在磁盘上的一组相关信息的集合，也称磁盘文件。为了区分不同的文件，可给每个磁盘文件一个标识，称为“磁盘文件名”。

1. 文件的分类

按照不同的分类方法，文件可以分成不同类。

(1) 文本文件与二进制文件

按文件中的数据格式分，文件可分成“二进制文件”和“文本文件”。

文本文件中存放的是与数据对应的字符的 ASCII 码，一个字符占一个字节。如有实数 -1234.567，要存放在文本文件中，将以字符 '-'、'1'、'2'、'3'、'4'、'.'、'5'、'6'、'7' 的形式存放，占 9 个字节。

二进制文件中的数据都是以二进制形式存放的，数据在文件中存放的格式和它在内存中的格式是一样的。如 -1234.567 可看成是一个 Real 型数据，在内存中占 4 个字节。

图 8-1 是存放三个相同实型数据的文本文件和二进制文件的内容（通过记事本打开）。

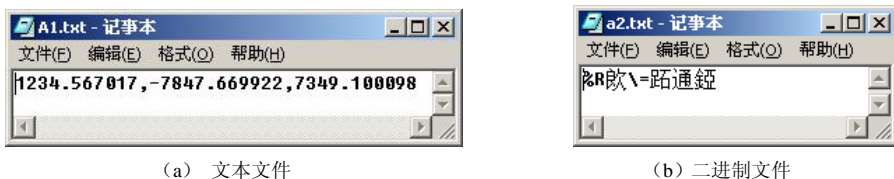


图 8-1 文本文件与二进制文件示例

可见，二进制形式存放数据占有存储空间较少，但不直观。文本文件占用的存储空间较多，比较直观。

(2) 顺序文件与随机文件

按文件的读写方式来分, 可以把文件分为“顺序文件”和“随机文件”。

对顺序文件来说, 读写必须从头开始。读取顺序文件中的数据时, 只能从第 1 个数据开始读取, 直到读取的数据是要处理的数据为止。如果要把处理后的这个数据写回顺序文件中, 也必须是从第 1 个数据开始, 依次把数据写到文件中, 当处理的这个数据已写回到数据文件后, 还必须继续读取并写回其后的所有数据。

顺序文件有点像录音带上的歌曲, 要听录音带上的第 5 首歌, 必须先“快进”过前面的 4 首歌, 相当于把前面的 4 首歌都读了一遍才能读第 5 首歌。同样, 要再录制录音带上的某一首歌, 为了不发生错误, 应从第 1 首歌开始依次录制所有的歌曲。

对随机文件来说, 读写的位置是任意的。只需利用系统函数将当前文件中的读写位置设置好, 就可以单独对这个数据进行读写操作。

随机文件有点像 CD 唱片, 要听第 4 首歌, 只需按一个 4 即可, 并不需要把前面的歌都读一遍。

另外把磁盘上的文件称为“磁盘文件”, 还可把输入输出设备看成“设备文件”。

2. 使用磁盘文件的步骤

磁盘文件是存放在磁盘上的, 而运行的程序只能处理在内存中的数据, 不能直接处理磁盘等外存储器上的数据, 因此只有把磁盘文件中的数据读到内存中, 才能在程序中使用。同样, 程序中产生或修改的都是内存中的数据, 必须把这些数据写到相应的文件中去, 才能使数据保存下来。

可见, 磁盘文件在使用之前必须“打开”, 在使用完毕后应要“关闭”。

下面给出了几种常见使用方式的实现步骤。

(1) 创建文件

创建一个不存在的文件, 一般要经历创建文件→写入数据→关闭文件几个步骤。其中创建文件的同时也就打开了文件。

(2) 读取文件

读写一个已存在的文件, 一般要经历读打开文件→读取数据→关闭文件几个步骤。

(3) 向一个文件的末尾添加数据

向一个已存在的文件末尾添加数据, 一般要经历添加打开文件→写入数据→关闭文件几个步骤。

(4) 读文件中某一位置的数据(随机读)

读取已存在文件的某个位置的数据, 一般要经历读打开文件→定位→读取数据→关闭文件几个步骤。

(5) 修改文件中某一位置的数据(随机写)

修改已存在文件的某个位置的数据, 一般要经历写打开文件→定位→修改数据→关闭文件几个步骤。

3. 文件指针

磁盘文件打开后, 将会产生一个指针, 它指向下一次要读写的数据位置, 该指针称为“文件指针”。文件指针具有自动移动的功能, 文件刚打开时, 文件指针指向磁盘文件中的第 1 个数据(记录或文本), 当读取了这个数据后, 文件指针自动指向下一个数据。当把数据写入某

个文件时，文件指针总是自动指向下一次要写入数据的位置。文件指针随文件的打开而存在，随文件的关闭而消失。文件指针与文件读写的关系如图 8-2 所示。

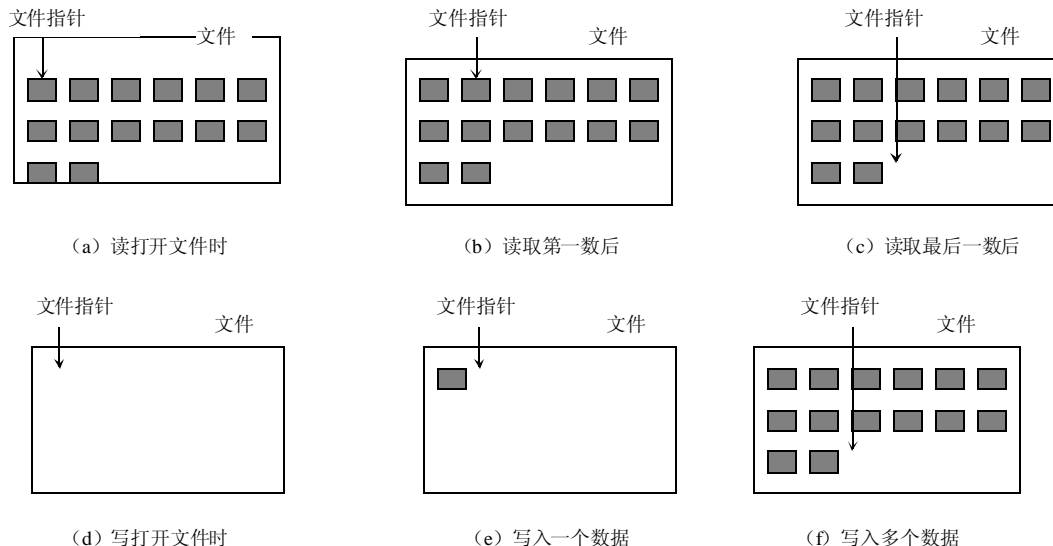


图 8-2 文件指针与数据读写（数据用带阴影的小方块组成）

8.1.2 Delphi 7 中的文件类型及文件类型变量的定义

在 Delphi 7 中，经常使用的文件有两类：**TextFile**（文本文件）和 **File**（二进制文件）。**TextFile** 文件中包含着一行或更多行能读懂的文本，**File** 包含了“其他一切东西”，存放形式是二进制，一般用来存放记录类型的数据（称记录型文件）。

1. 文本文件变量的定义

使用文件前需要定义一个文件型的变量，文本文件型变量的定义格式与功能如下。

[格式]：

Var

文本文件变量名: **TextFile**;

[功能]：定义一个文本文件变量。

例如，有下列语句：

var

LetterFile:TextFile;

该语句定义了一个 **TextFile** 类型的文本文件变量，变量名为 **LetterFile**。该变量可以与一个文本文件相联系，文本文件中可以用来存放字符。

2. 记录型文件变量的定义

记录型文件变量的定义一般要分成三个步骤，下面以定义一个存放学生数据的记录型文

件变量为例来说明记录型文件变量的定义方法。

(1) 定义记录类型

定义一个学生记录类型，如下：

```
type
  Student= record           //定义通讯记录型
    No:String[10];
    Name: string[10];
    Age:integer;
    Sex:String[2];
    Class:string[30];
  end;
```

(2) 定义记录文件类型

根据记录类型定义记录文件类型，定义记录文件类型语句的格式与功能如下。

[格式]:

记录文件类型名=File of 记录类型名

[功能]: 定义一个记录文件类型，类型名由“记录文件类型名”指定。

例如，有下列语句：

```
StudentFile = File of Student;
```

该语句定义了一个记录文件类型 StudentFile。

(3) 定义记录文件类型的变量

记录文件类型定义好后，应根据文件类型定义变量，根据文件类型定义变量的语句格式与功能如下。

[格式]:

```
var
```

记录文件类型变量名:记录文件类型;

[功能]: 根据记录文件类型定义记录文件类型变量。

例如，有下列语句：

```
var
```

```
VFStudent:StudentFile;
```

该语句定义了一个记录文件类型变量 VFStudent，该变量可以用来与一个记录文件相联系，相应文件中只能存放 Student 记录型的数据。

8.1.3 文本文件的使用

文本文件在使用之前必须要打开，对打开的文件可以进行读写操作，读写过后要关闭。

1. 文本文件的打开

打开文本文件可分成两个步骤：一是把文件变量与实际的文本文件关联起来；二是打开相应的文本文件。

(1) 关联文本文件变量与文本文件

可通过 AssignFile 过程来关联文件变量与实际的磁盘文件，该过程的格式与功能如下。

[格式]:

```
procedure AssignFile(var F: File; FileName: string);
```

[功能]: 把文件类型的变量 F 与参数 “FileName” 指定的磁盘文件关联起来。

[说明]: 关联了文件类型变量与磁盘文件过后，下面对磁盘文件的操作就可以通过文件型变量来进行了，程序中就不需要再与具体的磁盘文件打交道。

例如，有以下语句:

```
var  
    LetterFile: TextFile;           //定义文件型变量  
.....  
    AssignFile(LetterFile, 'MyLetter.txt'); //把文件型变量与文件名联系在一起
```

(2) 打开文本文件

把磁盘文件与文件型变量关联在一起后，就可以打开文件，打开文件可使用 Reset、Rewrite 或 Append 过程中的一个来进行。

Rewrite 过程的格式与功能如下。

[格式]:

```
procedure Rewrite(var F: File [; RecSize: Word ] );
```

[功能]: 以写方式创建与文件型变量 F 关联的文件。

[说明]: 参数 F 是文件型变量，参数 Word 是一个可选参数，当文件是一个无类型的文件时，该参数用来定义无类型文件的尺寸大小。如果磁盘上有一个同名的文件，该文件将被删除并创建一个新的空文件；如果指定的文件已经打开，将首先关闭该文件，然后再创建它，当前文件指针将置于空文件的开头。使用该种方法创建的文本文件只能用于写，不能读。

Append 过程的格式与功能如下。

[格式]:

```
procedure Append(var F: Text);
```

[功能]: 以添加方式打开与文件型变量 F 关联的文件。

[说明]: 参数 F 是一个文本文件型变量。如果与文件型变量 F 关联的文件不存在，将会产生一个错误；如果与文件型变量 F 关联的文件已经打开，将先把它关闭，然后再打开它。使用这种方式打开文件后，文件指针位于文件的结尾。

Reset 过程的格式与功能如下。

[格式]:

```
procedure Reset(var F [ : File; RecSize: Word ] );
```

[功能]: 以只读方式打开与文件型变量 F 相关联的文本文件。

[说明]: 如果打开的文件不存在，将会出现一个错误。如果文件已经打开，将首先关闭它再打开。打开过后文件位置指针位于文件的开头。

例如，有以下语句:

```
AssignFile(LetterFile, 'MyLetter.txt'); //把文件型变量与文件名联系在一起
```

```
Rewrite(LetterFile);           //创建指定的文件
```

2. 文本文件的关闭

关闭文件使用 **CloseFile** 过程，该过程的格式与功能如下。

[格式]:

```
procedure CloseFile(var F);
```

[功能]: 关闭与文件型变量 **F** 相关联的文件。

[说明]: 使用 **CloseFile** 过程关闭文件时，对文件的更新将实际写到磁盘文件中，并断开文件型变量 **F** 与实际的磁盘文件之间的联系。

例如，有下列语句:

```
AssignFile(LetterFile, MyLetter.txt);    //把文件型变量与文件名联系在一起
Rewrite(LetterFile);                    //创建指定的文件
.....
CloseFile(LetterFile);                  //关闭与文件型变量 LetterFile 联系在一起的文件
```

3. 文本文件的读写

(1) 文本文件的写操作

可以使用 **Write** 和 **WriteLn** 过程向文本文件中写入数据。**Write** 只是把指定数据写到文本文件中，**WriteLn** 不仅把数据写到文本文件中，而且还要在写入的数据后加上行结束标记。用这两个过程输出的数据可以是以下类型: **Char**、**Byte**、**Shortint**、**Word**、**Longint**、**Cardinal**、**Single**、**Real**、**Double**、**Extended**、**Currency**、**PChar**、**AnsiString**、**ShortString**、**string**、**Boolean** 和 **Bool**。

Write 过程的格式和功能如下。

[格式]:

```
procedure Write( [var F: Text; ] P1 [, P2,..., Pn] );
```

[功能]: 向文件型变量 **F** 关联的文件中写入 **P1**, **P2**, ..., **Pn** 等表达式的值。

[说明]: 参数 **F** 是一个文本文件变量，参数 **P1**, **P2**, ..., **Pn** 是要写入到文本文件中的表达式。

WriteLn 过程与 **Write** 过程的语法和功能基本相同，只是 **WriteLn** 过程在向文件中写入数据后自动在末尾添加一个换行符。

使用 **Write** 和 **WriteLn** 过程向文本文件中写入数据时，还可以指域宽和小数位。

要给 **Write** 和 **WriteLn** 过程输出的数据指定域宽，只需要相应的数据后增加一个 “:” 和一个表示域的宽度的数字。例如，有以下程序段:

```
var
  F1, F2: TextFile;           //定义两个文本文件变量
  N1, N2, N3: Integer;
  .....
  SFileName:= 'C:\A1.TXT';     //取得源文件名
  N1:=100; N2:=200; N3:=300;
  AssignFile(F1, SFileName);   //把源文件名和 F1 文件型变量联系在一起
  Rewrite(f1);
```



```

Writeln(F1,N1:10,N2:10,N3:10);
Writeln(F1,N1:2,N2:2,N3:2);
CloseFile(F1);           //关闭源文件
.....

```

在 C:\A1.TXT 文件中输出的结果如下:

```

100      200      300
100200300

```

可见域宽指定数据的输出宽度,当指定的宽度比数据的宽度小时,系统并不按照指定宽度截断数字,输出结果中所在数据挤在一起,各数据间没有间隔。

在把实数输出到文本文件中时,可以指定小数点位数。指定方法是在域宽数字后加“:”再加上表示小数点位数的整数。例如,有以下程序段:

```

var
  F1, F2: TextFile;           //定义两个文本文件变量
  N1,N2,N3:Real;
.....
SFileName:='C:\A1.TXT';      //取得源文件名
N1:=12.5;N2:=2.1;N3:=30.12;
AssignFile(F1, SFileName);    //把源文件名和 F1 文件型变量联系在一起
Rewrite(F1);
Writeln(F1,N1:10,N2:10,N3:10);
Writeln(F1,N1:10:2,N2:10:2,N3:10:2);
CloseFile(F1);               //关闭源文件
.....

```

在 C:\A1.TXT 文件中输出的结果如下:

```

1.2E+0001 2.1E+0000 3.0E+0001
12.50      2.10      30.12

```

可见如果不指定实数的小数位,实数将以指数形式输出。

(2) 文本文件的读操作

可使用 **Read** 和 **ReadLn** 过程从文本文件中读取数据,并赋值给相应的变量。**ReadLn** 和 **Read** 过程有一个不同之处,即该过程读取若干个数据后跳到下一行,而 **Read** 并不跳到下一行。

Read 过程的格式和功能如下。

[格式]:

```

procedure Read( [ var F: Text; ] V1 [, V2,...,Vn ] );

```

[功能]: 从文件型变量 **F** 关联的文件中读取 **n** 个数据并赋值给 **V1, V2, ..., Vn** 等变量。

[说明]: 如果读取的数据是数值型,数据之间可用空格隔开;如果读取的数据是 **String** 型,空格不能分隔数据,将一直读到换行符为止;为了读取指定数量的字符型,变量类型应该是 **String[N]** 型的。

例如, 假设 C:\a1.txt 文件中的数据如下:

12.50	2.10	30.10
Basic Pascal Delphi Visual Basic		

有下列程序段:

```
var
  F1, F2: TextFile;           //定义两个文本文件变量
  N1, N2, N3: Real;
  s1, s2, s3: string;
.....
  SFileName:= 'C:\A1.TXT';     //取得源文件名
  AssignFile(F1, SFileName);   //把源文件名和 F1 文件型变量联系在一起
  Reset(f1);
  Readln(F1, N1, N2, N3);      Read(F1, s1, s2, s3);
  Edit1.Text := FloatToStr(N1)+ ' ' + FloatToStr(N2)+ ' ' + FloatToStr(N3);
  Edit2.Text := S1; Edit3.Text := S2; Edit4.Text := S3;
  CloseFile(F1);               //关闭源文件
.....
```

程序运行后, 在 Edit1 中显示的数据为:

12.5 2.1 30.1

在 Edit2 中显示的数据为:

Basic Pascal Delphi Visual Basic

在 Edit3 和 Edit4 中显示的数据都为空。

4. 行尾与文件尾的判断

判断当前文件指针是否处于行尾和文件尾, 可通过函数 Eoln 和 Eof 函数来实现。

(1) Eof 函数

[格式]:

```
function Eof [ (var F: Text) ]: Boolean;
```

[功能]: 用于判断与文件型变量 F 相关联的文件的文件指针是否处于文件尾 (最后一个字符的后面), 如果处于文件尾, 则返回值为 True, 否则返回值为 False。需要注意的是, 如果文件为空, Eof 函数的返回值也是 True。

(2) Eoln 函数

[格式]:

```
function Eoln [(var F: Text) ]: Boolean;
```

[功能]: 用于判断与文件型变量 F 相关联的文件的文件指针是否处于行尾, 如果处于行尾, 则返回值为 True, 否则返回值为 False。

【例 8-1】 编写一个文本文件的复制程序。程序的设计界面如图 8-3 所示, 程序的运行界面如图 8-4 所示。程序运行时, 在【源文件名】编辑框中输入源文件名或通过单击【...】

按钮选择一个需复制的源文件名，在【目标文件名】编辑框中输入要复制成的目标文件名，然后单击【复制】按钮，将完成源文件到目标文件的复制。



图 8-3 程序设计界面



图 8-4 程序运行界面

【实现分析】

为实现把源文件复制成目标文件的功能，可首先定义两个单元级变量，分别用来存放源文件名和目标文件名（假设变量名为 SFileName 和 DFileName）。然后再定义两个文本文件型变量（假设变量名为 F1 和 F2），通过 AssignFile 过程把 F1 和源文件联系起来，把 F2 和目标文件联系起来。然后再使用 Reset 过程打开 F1 文件，使用 Rewrite 过程创建 F2 文件。为实现把源文件复制成目标文件的功能，可不断地从源文件中读取一个字符，把读取的字符写到目标文件中去。一直读取到文件尾为止，测试文件尾可用 Eof 函数来实现。文件复制过后应使用 CloseFile 过程关闭源文件和目标文件。

【界面设计】

本例的组件属性设置及组件功能如表 8-1 所示。

表 8-1 例 8-1 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'源文件名'	
Edit1	Text	"	用于输入和显示源文件名
Label2	Caption	'目标文件名'	
Edit2	Text	"	用于输入和显示目标文件名
Button1	Caption	'...'	单击它弹出一个对话框允许用户选择源文件名
Button2	Caption	'复制'	单击它把源文件复制成目标文件
Button3	Caption	'退出'	单击它将退出应用程序
OpenDialog1			打开对话框，用户选择源文件

【程序代码】

主要程序代码如下：

```
var
    SFilename,DFilename:string;           //存放源文件名和目标文件名的变量
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenDialog1.Execute ;                 //执行打开对话框选择源文件名
    Edit1.Text :=Opendialog1.FileName ;   //源文件名显示在 Edit1 编辑框中
```

```
end;
procedure TForm1.Button2Click(Sender: TObject);
var
  F1, F2: TextFile;           //定义两个文本文件变量
  Ch: Char;
begin
  SFileName:=Edit1.Text ;      //取得源文件名
  DFileName:=Edit2.text;       //取得目标文件名
  if length(SFilename)<>0 then  //如果源文件名不为空
  begin
    AssignFile(F1, SFileName);  //把源文件名和 F1 文件型变量联系在一起
    Reset(F1);                 //读打开源文件
    if length(DFilename)<>0 then //如果目标文件名不为空
    begin
      AssignFile(F2, DFileName); //把目标文件名和 F2 文件型变量联系在一起
      Rewrite(F2);               //创建打开目标文件
      while not Eof(F1) do       //文件指针不是位于文件的结尾
      begin
        Read(F1, Ch);           //从源文件中读取一个字符
        Write(F2, Ch);          //把读取的字符写到目标文件中去
      end;
      CloseFile(F2);            //关闭目标文件
    end
  else
    ShowMessage('目标文件名必须输入');
    CloseFile(F1);              //关闭源文件
    ShowMessage('文件复制完毕');
  end
else
  ShowMessage('源文件名必须输入');
end;
```

注意：程序运行时，输入正确的源文件名和目标文件名后单击【复制】按钮，然后可以在磁盘上找到相应的目标文件，打开它会发现与源文件的内容是一样的。

8.1.4 记录文件的使用

1. 记录型文件的打开和关闭

和文本文件一样，AssignFile 标准过程用来关联记录文件变量与实际的磁盘文件，Reset 和 Rewrite 过程用来打开记录文件，CloseFile 过程用来关闭记录文件。不能用 Append 过程来打开一个记录文件，它仅限于文本文件。打开文件前应调用 AssignFile 过程把文件名和特定文件变量关联起来，然后调用 Reset 或 Rewrite 过程来打开文件。Reset 过程为读或写打开文件，Rewrite 过程为读或写创建（或覆盖旧文件）一个新文件并打开它，如果试图用 Reset 过程打开一个不存在的文件，会发生运行时错误。

和文本文件相比，关闭文件的操作对记录文件更为重要。如果忘记关闭写过的文本文件，

最坏的情况是文件被截断，如最后一行写到文件中的数据并没真正的存入磁盘，这可能会导致一些麻烦，但一般来说很容易弥补。然而对于记录文件，如果在添加记录以后忘记关闭文件，可能会破坏整个文件。处理完文件后一定要关闭它——最好是刚处理完就关闭，至少应在退出程序前完成。

例如，有如下程序段：

```
var
    VFStudent:StudentFile;           //定义文件类型变量
    S1,S2:Student;
    .....
    AssignFile(VFStudent, 'C:\S1.DAT');
                                     //把记录型文件变量 VFStudent 与文件 C:\S1.DAT 相关联
    Rewrite(VFStudent);
    .....
    CloseFile(VFStudent);             //关闭与记录型文件变量 VFStudent 相关联的文件
```

2. 记录型文件的读写

要把记录写入到记录文件中，需调用 **Write** 函数并传递给它一个文件变量及一个或多个记录型变量。例如，下面的代码把变量 **s1** 和变量 **s2** 中的内容写入到与记录型文件变量 **VFStudent** 相关联的文件中去：

```
Write(VFStudent,S1); Write(VFStudent,S2);
```

上面的两条语句也可写成：

```
Write(VFStudent,S1,S2)
```

从记录型文件中读取数据可使用 **Read** 过程，可以一次读取一条记录到一个变量中，也可以一次读取多条记录存放到多个变量中。例如，下面的代码是从与记录型文件变量 **VFStudent** 相关联的文件中读取两条记录并存放到变量 **s1** 和变量 **s2** 中：

```
Read(VFStudent, S1,s2);
```

上面的语句也可写成：

```
Read(VFStudent,S1) ; Read(VFStudent, S2);
```

需注意的是，如果用 **Read** 过程读取记录，文件位置指针超过了文件尾，将会发生运行时错误。因此读取数据时一般要对文件尾进行判断。

3. 记录型文件的文件指针与记录数

可以认为文件有一个看不见的指针指向下一次 **Read** 或 **Write** 过程操作要作用的记录。无论何时打开文件，也不管是用 **Reset** 或 **Rewrite** 过程打开，这个不可见指针都定位在文件首。每次 **Read** 或 **Write** 操作都把文件指针前进一条记录使它从刚作用过的记录转移到下一个记录。如果读或写多个记录，对应于每一个被读或写的记录，文件指针都前进一次。

记录文件中的记录是顺序排列的，每条记录都有一个编号，记录编号从 0 开始。因此在包含 **N** 个记录的文件中，第一个记录的编号（或称记录号）是 0，最后一个记录的编号是 **N-1**。

如果想知道文件中有多少记录，可调用 **FileSize** 标准函数，该函数用来返回记录数。

4. 移动与检测记录文件的文件指针

移动和检测文件指针的位置使随机存取文件中的记录成为可能，记录的随机存取是通过 Seek 过程和 FilePos 函数来实现的。Seek 过程的作用是把文件指针移动到文件中的指定记录，而 FilePos 函数是返回文件指针当前所指的记录的记录号。

Seek 过程的格式和功能如下。

[格式]:

```
procedure Seek(var F; N: Longint);
```

[功能]: 把与记录型文件变量 F 关联的文件的文件指针移到编号为 N 的记录处，注意第一条记录的编号为 0。

FilePos 函数的格式和功能如下。

[格式]:

```
function FilePos(var F): Longint;
```

[功能]: 返回与记录型文件变量 F 关联的文件的文件指针所处的位置。

例如，要读取与 VFStudent 文件型变量相关联的文件的第五条记录，可使用如下语句：

```
Seek(VFStudent, 4);  
Read(VFStudent, S1);
```

注意：如果想读一个记录并修改它，然后再用新的信息覆盖旧信息，在写操作之前必须调用 Seek 函数以回到上一个记录。例如，下面的代码用来读取、修改并更新文件中的一条记录。

```
Seek(VFStudent,4);  
Read(VFStudent,S1);  
S1.Name='赵强';  
Seek(VFStudent,4);  
write(VFStudent,S1);
```

如果忘记第二次调用 Seek 过程，结果则将用更新后的信息覆盖了第 6 条记录。这是初学者常犯的错误。

5. 把记录文件的文件指针定位到文件尾的方法

如果把记录添加到记录文件的文件尾，在写操作之前，必须用 Seek 过程把文件指针移到文件的最后一条记录之后。最简单的办法是用 FileSize 获得文件的记录个数，并用这个数值作为 seek 的参数。

FileSize 函数的格式及功能如下。

[格式]:

```
function FileSize(var F): Integer;
```

[功能]: 返回与记录型文件变量 F 相关联的文件中包含的记录数。

例如，有下列语句：

```
Seek(VFStudent,FileSize(VFStudent));
```

由于记录数从 0 开始，在一个由 N 个记录组成的文件中最后一个记录数是 N-1。FileSize 返回的是文件的记录个数，值为 N 而不是 N-1。故上述语句刚好把文件指针定位到最后一个

记录的后面，此时执行写操作把记录写到文件中，也就是添加记录。

【例 8-2】 编写一个录入学生数据的应用程序，程序的设计界面如图 8-5 所示，程序的运行界面如图 8-6 所示。程序刚运行时，【录入】按钮可用，【确定】和【取消】按钮不可用。此时单击【录入】按钮，将清空所有的编辑框并等待用户输入一条新记录，并且设置【录入】按钮为不可用状态，设置【确定】和【取消】按钮为可用状态。当用户输入了一条记录后单击【确定】按钮将把输入的记录写入到文件中，单击【取消】按钮将取消用户的输入，并把最后一条记录的值显示在界面上，无论是单击【确定】按钮还是【取消】按钮后，均把【录入】按钮设置为可用状态，把【确定】和【取消】按钮设置为不可用状态，且把所有的编辑框置为只读状态。



图 8-5 程序设计界面

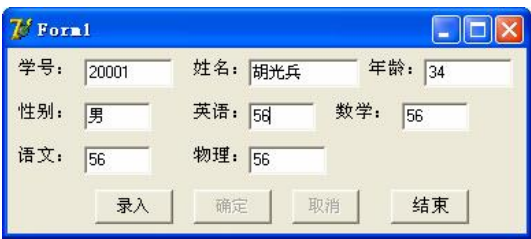


图 8-6 程序运行界面

【实现分析】

为在程序中能够处理学生信息，可定义一个能够反映学生信息的记录类型，再根据该类型定义记录变量。为存放学生的数据，可定义一个记录型文件。在程序刚运行时，打开记录文件，并把最后一条记录的信息显示出来。当用户录入一条记录后，单击【确定】按钮将该记录添加到文件的尾部，单击【取消】按钮将取消用户的输入并把最后一条记录的信息显示在界面上。为了能够方便地把用户输入的信息转换成记录型数据和把记录型数据显示在界面上，本例定义了两个过程 SetRecord 和 DispRecord，分别用来实现这两个功能。

【界面设计】

本例的组件属性设置及组件功能如表 8-2 所示。

表 8-2 例 8-2 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'学号'	
Edit1	Text	"	输入和显示学生学号信息
Label2	Caption	'姓名'	
Edit2	Text	"	输入和显示学生姓名信息
Label3	Caption	'年龄'	
Edit3	Text	"	输入和显示学生年龄信息
Label4	Caption	'性别'	
Edit4	Text	"	输入和显示学生性别信息
Label5	Caption	'英语'	
Edit5	Text	"	输入和显示学生英语成绩信息

续表

组 件 名	属 性 名	属 性 值	作 用
Label6	Caption	'数学'	
Edit6	Text	"	输入和显示学生数学成绩信息
Label7	Caption	'语文'	
Edit7	Text	"	输入和显示学生语文成绩信息
Label8	Caption	'物理'	
Edit8	Text	"	输入和显示学生物理成绩信息
Button1	Caption	'录入'	录入一条学生记录
Button2	Caption	'确定'	把一条学生记录写到文件中去
Button3	Caption	'取消'	取消刚才所做的录入操作
Button4	Caption	'退出'	关闭文件并退出应用程序

【程序代码】

```

var
  Form1: TForm1;
implementation
type
  Student= record                                //定义存放学生信息的记录型
    No:String[10];      Name: string[10];
    Age:integer;        Sex:String[2];
    English:Real;       Math:Real;
    Chinese:Real;       Physics:Real;
    Aver:Real;
  end;
  StudentFile = file of Student;                //定义记录文件类型
var
  VFStudent:StudentFile;                        //定义文件类型变量
  Stu:Student;                                  //定义记录型变量
  CurRecNo:Integer;                             //该变量用来记录当前记录号
{$R *.dfm}
Procedure EditClear();                          //清除各编辑框中的内容
begin
  Form1.Edit1.text:="";
  .....
  Form1.Edit8.text:="";
end;
Procedure EditReadOnly();                       //使各编辑框只能读
begin
  Form1.Edit1.ReadOnly:=True;
  .....
  Form1.Edit8.ReadOnly:=True;
end;
Procedure EditReadWrite();                     //使各编辑框可读可写

```



```

begin
    Form1.Edit1.ReadOnly:=False;

    .....

    Form1.Edit8.ReadOnly:=False;
end;
procedure DispRecord(StuRec:Student);           //把记录型变量 StuRec 各分量的值显示出来
begin
    Form1.Edit1.Text :=StuRec.No ;
    Form1.Edit2.Text :=StuRec.Name;
    Form1.Edit3.Text :=inttostr(StuRec.Age) ;
    Form1.Edit4.Text :=StuRec.Sex;
    Form1.Edit5.Text :=floattostr(StuRec.English ) ;
    Form1.Edit6.Text :=floattostr(StuRec.Math) ;
    Form1.Edit7.Text :=floattostr(StuRec.Chinese) ;
    Form1.Edit8.Text :=floattostr(StuRec.Physics) ;
end;
procedure SetRecord(Var StuRec:Student);        //把编辑框中的内容存放到记录型变量 StuRec 中
begin
    StuRec.No :=Form1.Edit1.Text ;
    StuRec.Name:=Form1.Edit2.Text ;
    StuRec.Age:=strtoint(Form1.Edit3.Text);
    StuRec.Sex:=Form1.Edit4.Text ;
    StuRec.English:= strtofloat(Form1.Edit5.Text);
    StuRec.Math:=Strtofloat(Form1.Edit6.Text);
    StuRec.Chinese:=Strtofloat(Form1.Edit7.Text);
    StuRec.Physics:=Strtofloat(Form1.Edit8.Text);
end;
procedure TForm1.FormCreate(Sender: TObject);
Var
    CurDir:String;
    StuFilename:String;
    M:integer;
begin
    getdir(0,CurDir);
    StuFileName:=CurDir+'\Student.dat';           //形成存放学生信息的文件名
    AssignFile(VFStudent,StuFileName);           //把文件型变量与物理文件关联在一起
    If NOT FileExists(StuFileName) then           //如果文件不存在
        Rewrite(VFStudent)                       //创建文件
    Else
        Reset(VFStudent);                       //否则打开文件
    If Eof(VFStudent) then                       //如果没有记录
        EditClear                               //清除界面上显示的内容
    else
        begin                                    //如果有记录
            M:=filesize(VFStudent);              //得到记录数

```

```

        if m=0 then                                //如果文件为空
            EditClear                               //清除编辑框
        else
            begin
                seek(VFStudent,M-1);                //定位到最后一条记录
                Read(VFStudent,stu);                 //读取最后一条记录
                DispRecord(stu);                     //显示该记录的值
            end;
        end;
    end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Button1.Enabled :=False;                        //使录入按钮不可用
    Button2.Enabled :=True;                         //使确定按钮可用
    Button3.Enabled:=True;                          //使取消按钮可用
    EditClear;                                       //清除编辑框的内容
    EditReadWrite();                                //使编辑框可读可写
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    M:integer;
begin
    Button1.Enabled :=True;                        //使录入按钮可用
    Button2.Enabled :=False;                       //使确定按钮不可用
    Button3.Enabled:=False;                        //使取消按钮不可用
    EditReadOnly;                                  //使编辑框只读
    M:=filesize(VFStudent);                        //求得记录数
    seek(VFStudent,M);                             //定位到文件尾
    SetRecord(stu);                                //把输入的数据写得记录变量中去
    Write(VFStudent,stu);                          //把记录写到文件中
end;
procedure TForm1.Button3Click(Sender: TObject);
Var
    M:integer;
begin
    Button1.Enabled :=True;                        //使录入按钮可用
    Button2.Enabled :=False;                       //使确定按钮不可用
    Button3.Enabled:=False;                        //使取消按钮不可用
    EditReadOnly;                                  //使编辑框只读
    M:=filesize(VFStudent);                        //得到记录数
    if m=0 then                                    //如果文件为空
        EditClear                                  //清除编辑框
    else
        begin
            seek(VFStudent,M-1); Read(VFStudent,stu); //读取最后一条记录
        end
    end
end;

```

```
DispRecord(stu);           //显示该记录的值
end;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    CloseFile(VFStudent);Application.Terminate;           //关闭文件，程序结束
end;
```

8.2 典型实例

8.2.1 典型实例一

【实例题目】：文本文件的显示、修改和保存

编写显示、修改和保存一个文本文件的程序，程序的设计界面如图 8-7 所示，程序的运行界面如图 8-8 所示。程序运行时，在文件名后面的编辑框中可以输入要打开的文件名，也可以通过单击其后的【...】按钮来弹出一个打开对话框供用户选择一个文本文件；单击【打开】按钮，将打开相应的文件并把文件的内容显示在 Memo 组件中；若在程序运行过程中修改了 Memo 组件中显示的文本内容，然后单击【保存】按钮将把文件保存起来；单击【关闭】按钮将关闭文件和结束应用程序。（要求：不得通过 Memo1.Lines 的 SaveToFile 和 LoadFromFile 方法来打开文件和保存文件）



图 8-7 程序设计界面



图 8-8 程序运行界面

【实现方法】

在【打开】按钮的 OnClick 事件中，打开相应的文本文件，并通过一个循环从文本文件中一行一行地读取数据并添加到 TMemo 组件的 Lines 属性中，循环的结束条件是到了文件尾。保存时，可使用 Rewrite 方法重新打开文件，然后通过一个 For 循环把 TMemo 组件中显示的文本一行一行地写入到文件中。

【界面设计】

本例的组件属性设置及组件功能如表 8-3 所示。

表 8-3 典型实例—组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'文件名: '	
Edit1	Text	"	输入和显示要打开的文本文件名
OpenDialog1	Filter	'文本文件*.txt'	弹出打开对话框让用户选择一个文本文件
Button1	Caption	'...'	单击它将弹出打开对话框
Button2	Caption	'打开'	单击它将打开文件
Button3	Caption	'保存'	单击它将保存文件
Button4	Caption	'关闭'	单击它将关闭文件并结束程序的运行

【程序代码】

```

implementation
var
    F1: TextFile;           //定义一个文本文件变量
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog.Execute ;           //弹出打开对话框
    edit1.Text :=OpenDialog1.FileName; //显示要打开的文件名
end;
procedure TForm1.Button2Click(Sender: TObject); //打开文件并把文件内容显示在 Memo1 中
var
    Line:string;
begin
    if length(Edit1.Text)<>0 then           //输入了文件名
    begin
        AssignFile(F1,Edit1.Text );
        Reset(F1);
        Memo1.Lines.Clear ;
        while not eof(F1) do               //不是文件的结尾
        begin
            readln(F1,line);               //读取一行
            Memo1.Lines.Add(Line);         //把该行显示在 Memo1 中
        end;
    end
    else
        ShowMessage('必须输入文件名');
end;
procedure TForm1.Button3Click(Sender: TObject);
var
    i:integer;
begin

```

```
if Memo1.Modified then           //如果 Memo1 中的内容已经修改
begin
    Rewrite(f1);                  //创建打开文件
    for i:=0 to Memo1.Lines.Count -1 do //把 Memo1 中的内容写到文件中
        Writeln(f1,Memo1.Lines[i]);
    end;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    CloseFile(f1);Application.Terminate; //关闭文件
end;
```

8.2.2 典型实例二

【实例题目】：求学生的平均成绩并保存

在例 8-2 的基础上添加求学生平均成绩的功能。程序的设计界面如图 8-9 所示，程序的运行界面如图 8-10 所示。程序运行时，单击**【求平均成绩】**按钮将求出所有学生的平均成绩，然后在**【记录号】**编辑框中输入一个记录号，单击**【显示】**按钮将显示相应记录的学生信息，包括平均分。

图 8-9 程序设计界面

图 8-10 程序运行界面

【实现方法】

为求所有学生的平均成绩，可通过一个循环，依次从学生数据文件中读取学生数据，求得学生的平均成绩并赋值给它的 **Aver** 域，再把修改后的数据写到学生数据文件中。因此对于每条记录，首先要读出来，处理完后应要写回文件中。由于读出数据后，文件指针指向的记录已不再是该记录，而是下一条记录，因此在写入之前，还必须把文件指针定位到要写入的位置，再写入。

【界面设计】

本例在例 8-2 的基础上增加了一些组件，增加的组件属性设置及组件功能如表 8-4 所示。

表 8-4 典型实例二增加的组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
Label9	Caption	'平均分: '	
Edit9	Text	"	显示学生的平均分信息
Label10	Caption	'记录号: '	
Edit10	Text	"	输入要显示的学生的记录号
Button5	Caption	'求平均成绩'	单击它将求出所有学生的平均成绩
Button6	Caption	'显示'	单击它将显示相应记录号的学生信息

【程序代码】

在例 8-2 基础上增加的程序代码如下:

```

procedure DispRecord(StuRec:Student);           //把记录型变量 StuRec 各分量的值显示出来
begin
    .....
    Form1.Edit9.Text :=floattostr(StuRec.Aver);   //显示平均成绩
end;
procedure TForm1.Button5Click(Sender: TObject);
var
    M:integer;                                   //要修改记录的记录号
begin
    Seek(VFStudent,0);M:=0;
    While Not Eof(VFStudent) do                 //当不是文件尾
    begin
        Read(VFStudent,Stu);                   //读当前记录
        Stu.Aver:=(Stu.English+stu.Math +stu.Chinese +stu.Physics)/4;   //求平均成绩
        Seek(VFStudent,M);                     //定位到刚才读出的记录
        Write(VFStudent,stu);                  //写入文件
        M:=M+1;                                // 记录号加 1
    end;
end;
procedure TForm1.Button6Click(Sender: TObject);
begin
    SEEK(VFStudent,Strtoint(edit10.text));       //定位到指定记录
    Read(VFStudent,stu);                         //读取一条记录
    DispRecord(Stu);                             //显示该记录
end;

```

8.3 上机练习

8.3.1 上机练习一

【练习题目】：统计文本文件中的单词数

编写一个统计文本文件中的单词数的程序。程序的设计界面如图 8-11 所示，程序的运行界面如图 8-12 所示。程序运行时，可在【文件名】编辑框中输入一个文本文件名或通过打开对话框选择一个文本文件名，然后单击【统计】按钮将统计出该文本文件中包含的单词个数

并显示在第二个编辑框中。单击【退出】按钮将关闭文件并退出应用程序。

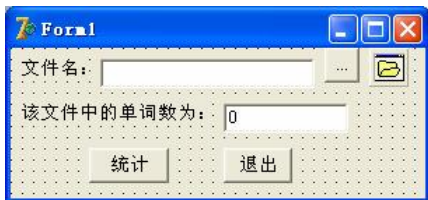


图 8-11 程序设计界面

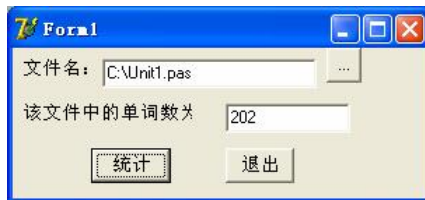


图 8-12 程序运行界面

【要点提示】

本题的重点是单词数目的统计。单词数目可以由空白字符（空格、Tab 键和 Enter 键）出现的次数决定（连续出现的若干个空白字符作为一个空白字符，一行开头的空白字符不统计在内）。如果测出某一个字符为非空白字符，而它的前面的字符是空白字符，则表示“新的单词开始了”，此时应使单词数加 1。如果当前字符为非空白字符而其前面的字符也是非空白字符，则意味着该字符仍然是原来那个单词的继续，单词数不再加 1。前面一个字符是否为空白字符可以用一个变量 flag 来表示，若 flag 等于 0，则表示前一个字符是空白字符；如果 flag 等于 1，则表示前一个字符为非空白字符。

【参考代码】

```
implementation
var
    F1: TextFile;           //定义一个文本文件变量
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog1.Execute ;           //弹出打开对话框
    edit1.Text :=OpenDialog1.FileName;   //显示要打开的文件名
end;
procedure TForm1.Button2Click(Sender: TObject); //打开文件并统计单词数
var
    ch:char;                 //存放读取的字符
    n,flag:integer;          //存放字符个数和标记是否为空白字符
begin
    if length(Edit1.Text)<>0 then       //输入了文件名
    begin
        AssignFile(F1,Edit1.Text );
        Reset(F1);
        n:=0; flag:=0;
        while not eof(F1) do           //不是文件的结尾
        begin
            read(F1,ch);               //读取一个字符
            if (ch=' ') Or (ord(ch)=13) Or (Ord(ch)= 9) then
                flag:=0                //如果是空白字符，则标记 Flag 的值置 0
            else
```

```

        if flag=0 then           //如果不是空白字符，且前面是空白字符
        begin
            flag:=1;             //标志 Flag 置 1，表示该字符不是空白字符
            n:=n+1;              //单词数加 1
        end;
    end;
    Edit2.Text :=Inttostr(n);
end
else
    ShowMessage('必须输入文件名');
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    CloseFile(f1);Application.Terminate;    //关闭文件，退出程序
end;

```

8.3.2 上机练习二

【练习题目】：学生信息的浏览

编写一个学生信息的浏览程序，程序的设计界面如图 8-13 所示，程序的运行界面如图 8-14 所示。程序运行时，单击【首记录】按钮将会显示第一条记录的内容，单击【前移】按钮将会显示前一条记录的内容，单击【后移】按钮将会显示下一条记录的内容，单击【末记录】按钮将会显示最后一条记录的内容。注意移动记录指针的时候，相应按钮的状态变化，如移动到文件头时，【首记录】和【前移】按钮不能用，移动到文件尾时，【后移】和【末记录】按钮不能用。



图 8-13 程序设计界面



图 8-14 程序运行界面

【要点提示】

本题的关键是确定要显示记录的记录号，可用一个变量 `CurRecNo` 来存放当前要显示的记录号。程序刚运行时，显示第一条记录，`CurRecNo` 值为 1。当单击【首记录】按钮时，把 `CurRecNo` 的值重新设置为 1，同时把记录号为 `CurRecNo` 的记录显示出来。当单击【前移】按钮时，`CurRecNo` 的值减 1，并且把记录号为 `CurRecNo` 的记录显示出来，这时将涉及文件头的判断，当 `CurRecNo` 的值小于 1 时，到达文件头，应把记录号重新设为 1。当单击【后移】按钮时，`CurRecNo` 的值应加 1，同时把记录号为 `CurRecNo` 的记录显示出来，这时将涉及文件尾的判断，当 `CurRecNo` 的值大于记录数（通过 `FileSize` 函数来获得）时，到达文件尾，此

时应把 CurRecNo 的值设为记录数。当单击【未记录】时，把 CurRecNo 的值重新设置为记录数，同时把记录号为 CurRecNo 的记录显示出来。

【参考代码】

在典型实例二的基础上删除所有按钮，在 Form1 的 OnCreate 事件过程的末尾增加一条语句：

```
CurRecNo:=1;
```

增加的 4 个按钮的 OnClick 事件代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    CurRecNo:=1;                //记录号为 1
    Seek(VFStudent, CurRecNo-1); //定位到第一条记录
    Read(VFStudent, Stu);       //读取第一条记录内容
    DispRecord(Stu);            //显示第一条记录的值
    Button1.Enabled := False;    //以下 4 条语句是按钮状态变化
    Button2.Enabled := False;
    Button3.Enabled := True;
    Button4.Enabled := True;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    CurRecNo:=CurRecNo-1;      //记录号减 1
    Button3.Enabled := True;    //以下两条语句是按钮状态变化
    Button4.Enabled := True;
    if CurRecNo>=1 then        //如果不是文件头
    begin
        Seek(VFStudent, CurRecNo-1); //定位到该记录
        Read(VFStudent, Stu);       //读取该记录
    end
    else
    begin
        //已到文件头
        ShowMessage('已到文件头');
        CurRecNo:=1;                //重新设置记录号为 1
        Button1.Enabled := False;    //以下两条语句为按钮状态变化
        Button2.Enabled := False;
    end;
    DispRecord(Stu);            //显示当前记录
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    CurRecNo:=CurRecNo+1;      //记录号加 1
    Button1.Enabled := True;     //以下两条语句是按钮状态变化
    Button2.Enabled := True;
    if CurRecNo<=FileSize(VFStudent) then //如果没有到达文件尾
```

```
begin
    Seek(VFStudent, CurRecNo-1);           //定位到该记录
    Read(VFStudent, Stu);                  //读取该记录
    DispRecord(Stu);                       //显示该记录内容
end
else
begin
    ShowMessage('已到文件尾');
    CurRecNo:=FileSize(VFStudent);         //重新定位到文件尾
    Button3.Enabled :=False;   Button4.Enabled :=False;
    DispRecord(Stu);                       //显示该记录内容
end;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    CurRecNo:=FileSize(VFStudent);         //获得最后一条记录的记录号
    Seek(VFStudent, CurRecNo-1);           //定位到最后一条记录
    Read(VFStudent, stu);                  //读取最后一条记录的内容
    DispRecord(stu);                       //显示该记录的值
    Button1.Enabled :=True;                //以下 4 条语句为按钮状态变化
    Button2.Enabled :=True;
    Button3.Enabled :=False;   Button4.Enabled :=False;
end;
```

课后考场

一、选择题（20 分，每题 5 分）

- 关于文件指针的说法，下列不正确的是_____。
 - 文件指针指向下一个要写入数据的位置
 - 文件指针指向下一个要读取数据的位置
 - 文本文件没有文件指针
 - 二进制文件一打开，将会自动产生一个文件指针
- 某文本文件已经存在，现在要读打开它，应使用_____过程。
 - Rewrite
 - Append
 - Reset
 - AssignFile
- 记录型文件的存取是以记录作为单位进行的，记录型文件能够进行随机存取，为进行随机存取，应首先把文件指针定位到要读取的记录位置，此操作使用的过程是_____。
 - Reset
 - Seek
 - FilePos
 - FileSize
- 下列的_____过程或函数不能用于记录型文件。
 - Rewrite
 - Append
 - Reset
 - AssignFile

二、填空题（40 分，每空 5 分）

- 文件在使用前要_____，文件在使用后应_____。
- 已知 F1 为文本类型的文件型变量，现在要在 F1 关联的文件的末尾添加信息，使用的

打开文件的语句应是_____。

3. 文件在使用之后要关闭, 关闭文件型变量 F1 关联的文件的语句是_____。

4. 要返回与记录型文件变量 F1 关联的文件中包含的记录数, 使用的函数调用为_____。

5. 把与记录型文件变量 F1 关联的文件的文件指针移到最后一条记录的后面, 应执行的语句是_____。

6. 记录文件中的记录是顺序排列的, 每条记录都有一个编号, 记录编号从_____开始。

7. 要判断与记录型文件变量 F1 关联的文件的文件指针是否位于文件尾, 应使用的函数调用为_____。

三、程序设计题 (40 分)

编写一个通讯录的输入、读取和修改的程序, 程序的设计界面如图 8-15 所示, 程序的运行界面如图 8-16 所示。程序刚运行时, 用户无法输入信息。用户单击【录入】按钮后, 可以输入一条通讯录信息, 同时【录入】和【读取且修改】按钮不可用, 【存盘】和【取消】按钮可用。此时单击【存盘】按钮, 允许用户输入, 输入的一条通讯录信息将存入到相应的文件中, 单击【取消】按钮, 将取消用户输入的一条通讯录信息, 无论单击【存盘】还是【取消】按钮, 均会使【录入】和【读取且修改】按钮可用, 【存盘】和【取消】按钮不可用, 且不允许用户输入。当单击【读取且修改】按钮时, 将弹出一个对话框要求用户输入一个记录号, 然后把该记录号的记录内容显示在界面上让用户修改, 界面上按钮及组件的状态变化和单击【录入】按钮完全一致。

图 8-15 程序设计界面

图 8-16 程序运行界面

第 9 章 应用程序界面设计技术

本章要点


- ▮ 多窗体程序设计
- ▮ SDI 应用程序设计
- ▮ MDI 应用程序设计
- ▮ 变量的作用域


9.1 理论知识

9.1.1 多窗体程序的设计

很少有应用程序只由一个窗体组成，很多应用程序都是由多个窗体组成，并且根据需要可以在窗体之间切换。

1. 为应用程序添加和删除窗体

新建一个 Delphi 项目时，项目中只有一个名为 Form1 的窗体，如果要为当前应用程序添加窗体，可通过单击工具栏上的新建窗体  按钮来实现。每单击一次该按钮将为应用程序添加一个新窗体。

如果想去除当前项目中的某窗体，可通过单击工具栏上的移去文件  按钮来实现，单击该按钮后，将会出现如图 9-1 所示的【Remove From Project】（移去文件）对话框，在该对话框中选中要移去的窗体，然后单击【OK】按钮，将会出现如图 9-2 所示的【Confirm】（确认）对话框，单击【Yes】将把相应的窗体或单元文件移去。

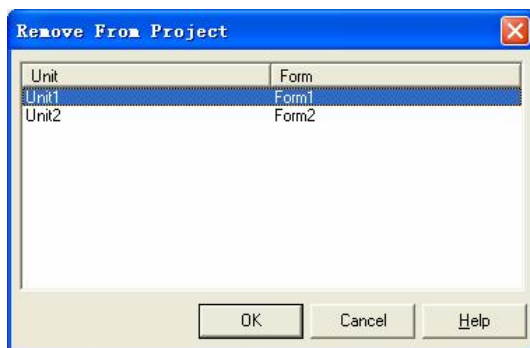



图 9-1 【Remove From Project】对话框



图 9-2 【Confirm】对话框

2. 窗体间切换

当应用程序中有很多窗体时,在编辑状态下对某一窗体进行修改就要进行窗体间的切换,切换方法是单击工具栏上的查看窗体按钮,将会出现如图 9-3 所示的【View Form】(查看窗体)对话框,在该对话框中选中需要查看或修改的窗体后单击【OK】按钮,选中的窗体将出现在最前面,供用户查看和修改。

3. 设置主窗体和自动创建窗体

所谓主窗体是指程序开始运行时出现的窗体,新建一个 Delphi 项目时,第一个窗体默认为主窗体,但主窗体可以重新设定。设定方法是执行【Project】→【Options】命令,将出现如图 9-4 所示的【Project Options for Project1.exe】(项目选项)对话框,在该对话框的【Main Form】列表框中可以选择要设置为主窗体的窗体。

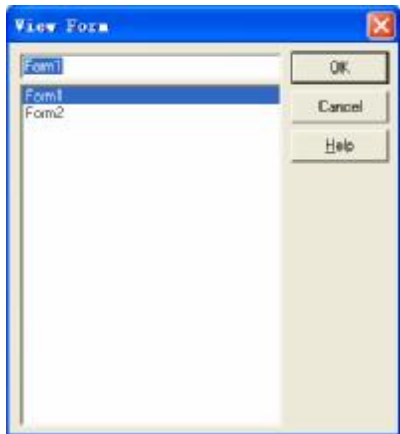


图 9-3 【View Form】对话框

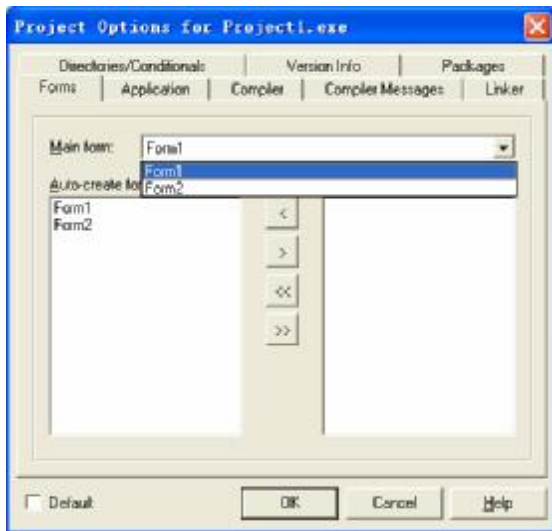


图 9-4 【Project Options for Project1.exe】对话框

【例 9-1】 编写一个调查用户购买商品的应用程序。程序运行时将出现一个调查窗体供用户输入信息,如图 9-5 所示。用户输入信息后单击【提交】按钮将出现确认窗体,显示用户输入的信息,如图 9-6 所示。在该窗体上,用户单击【确认】按钮将退出应用程序,如果用户单击【重填】按钮将又回到图 9-5 所示的【调查窗体】等待用户输入信息。

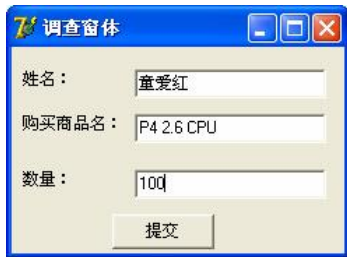


图 9-5 程序运行界面 (一)

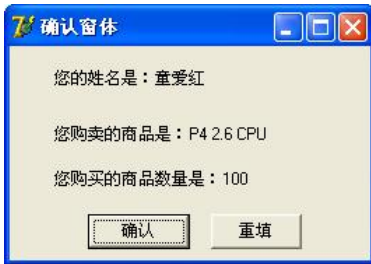


图 9-6 程序运行界面 (二)

【实现分析】

创建项目时，把 Form1 作为调查窗体，再添加一个窗体 Form2 作为确认窗体。在 Form1 窗体上单击【提交】按钮时，可把 Form1 窗体隐藏起来（调用 Hide 方法），然后把 Form2 窗体显示出来（调用 Show 方法）。在 Form2 窗体中用三个 Label 组件显示用户输入的信息，可在 Form2 窗体的 Show 事件中把用户输入的信息显示在这三个 Label 组件中。在 Form2 窗体中单击【确认】按钮时要结束程序的运行，只需把两个窗体均关闭即可（调用 Close 方法），在 Delphi 中关闭了主窗体也就结束了程序的运行。

【界面设计】

新建一个项目，按表 9-1 为 Form1 窗体添加组件并设置组件的属性。

表 9-1 例 9-1 的 Fom1 窗体上组件对象及其属性设置

对 象 名	属 性 名	属 性 值	说 明
Form1	Caption	'调查窗体'	用来接收用户输入的信息
Label1	Caption	'姓名: '	
Edit1	Text	"	输入姓名
Label2	Caption	'购买商品名: '	
Edit2	Text	"	输入购买的商品名称
Label3	Caption	'数量: '	
Edit3	Text	"	输入购买的商品数量
Button1	Caption	'提交'	供用户对输入的数据进行提交


设置好 Form1 窗体后，单击新建窗体  按钮为项目添加一个名为 Form2 的窗体，按表 9-2 所示为 Form2 窗体添加组件并设置组件的属性。

表 9-2 例 9-1 的 Fom2 窗体上组件对象及其属性设置

对 象 名	属 性 名	属 性 值	说 明
Form2	Caption	'确认窗体'	显示用户输入的信息供确认
Label1	Caption	"	显示用户输入的姓名信息
Label2	Caption	"	显示用户输入的购买商品名称信息
Label3	Caption	"	显示用户输入的购买数量信息
Button1	Caption	'确认'	用户单击该按钮时退出应用程序
Button2	Caption	'重填'	用户单击该按钮时将返回 Form1，以便重新输入数据

【程序代码】

Form1 窗体的程序代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Form1.Hide;
    Form2.show;
end;
```

Form2 窗体的程序代码如下:

```
procedure TForm2.FormShow(Sender: TObject);
begin
    Label1.Caption := '您的姓名是: ' + Form1.Edit1.Text;
    Label2.Caption := '您购买的商品是: ' + Form1.Edit2.Text;
    Label3.Caption := '您购买的商品数量是: ' + Form1.Edit3.Text ;
end;
procedure TForm2.Button1Click(Sender: TObject);
begin
    Form2.Close ;
    Form1.Close ;
end;
procedure TForm2.Button2Click(Sender: TObject);
begin
    Form2.Hide;
    Form1.Show;
end;
```

注意: 代码输入好后, 执行程序, 将会出现如图 9-7 所示的提示信息。原因是在 Form1 中引用了 Form2 窗体, 需把 Form2 窗体单元加载到 Form1 的 Uses 列表中, 单击【Yes】按钮即可。

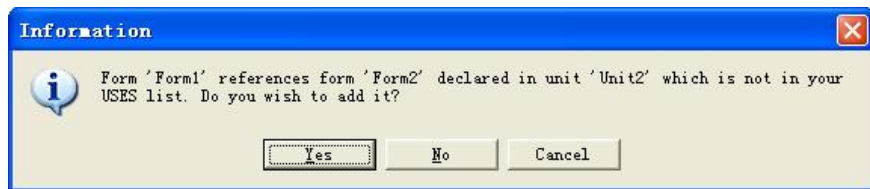


图 9-7 提示信息

9.1.2 SDI 应用程序设计技术

Windows 附件中的记事本程序一次只能打开一个文档。当一个文档打开后, 如果想要打开其他的文档, 必须先关闭已打开的文档。像记事本程序这样的一次只允许打开一个文档的应用程序称单文档界面 (Single Document Interface, SDI) 应用程序, SDI 应用程序比较简单, 可以使用 Delphi 提供的模板来设计, 也可以由用户直接从头开始设计。本节只介绍使用模板创建 SDI 的方法, 由用户从头开始设计 SDI 应用程序的方法可参见本章后面的“典型实例一”。

【例 9-2】 利用 Delphi 提供的模板创建一个标准的 SDI 应用程序。

创建步骤如下。

(1) 执行【File】→【New】→【Other】命令, 将会出现如图 9-8 所示的【New Item】对话框, 在该对话框中选中【Projects】选项卡, 再选中图标【SDI Application】。单击【OK】按钮, 将会出现如图 9-9 所示的【Select Directory】对话框。

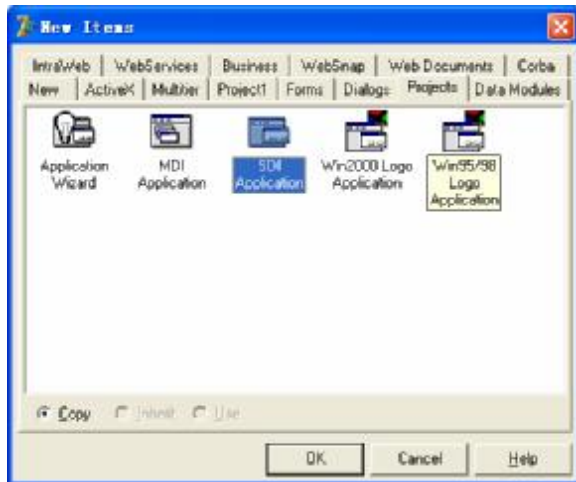


图 9-8 【New Items】对话框

(2) 在【Select Directory】对话框中选择 SDI 应用程序的保存路径，然后单击【OK】按钮，系统将自动创建一个标准的单文档应用程序，如图 9-10 所示。

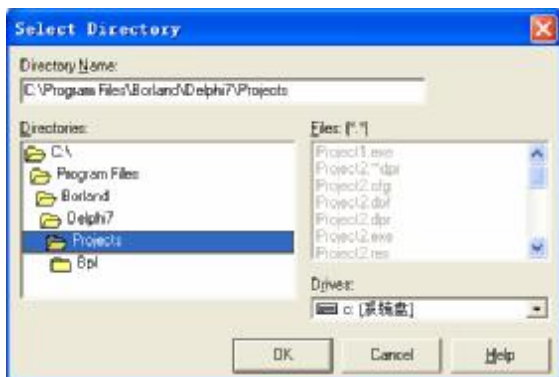


图 9-9 【Select Directory】对话框



图 9-10 系统创建的 SDI 应用程序

创建的 SDI 应用程序只是一个模板，可以向其中增加组件，修改程序代码来实现自己想要实现的功能。

9.1.3 MDI 应用程序设计技术

1. MDI 应用程序的概念

与 SDI 应用程序相对应，MDI 应用程序是允许在一个主窗体中创建多个子窗口的应用程序。例如 Microsoft Excel 与 Microsoft Word for Windows 等就是具有多文档界面（Multiple Document Interface, MDI）的应用程序。MDI 应用程序允许用户同时显示多个文档，每个文档显示在它自己的窗口中。文档或子窗口被包含在父窗口中，父窗口为应用程序中所有子窗口提供工作空间。子窗体就是普通窗体。一个应用程序可以包含许多相似或者不同样式的子窗体。在运行时，子窗体显示在父窗体工作空间之内，且不能移出父窗体。当子窗体最小化

时，它的图标显示在父窗体的工作空间之内，而不是在任务栏中。

创建 MDI 应用程序，可以通过模板也可以由用户从头开始创建。

2. 通过模板创建 MDI 应用程序

【例 9-3】 利用 Delphi 提供的模板创建一个标准的 MDI 应用程序。

应用模板创建 MDI 应用程序的步骤如下。

(1) 执行【File】→【New】→【Other】命令，将会出现如图 9-8 所示的【New Item】对话框，在该对话框中选中【Projects】选项卡，再选中图标【MDI Application】，然后单击【OK】按钮。将会出现如图 9-9 所示的【Select Directory】对话框。

(2) 在【Select Directory】对话框中选择 MDI 应用程序的保存路径，然后单击【OK】按钮，系统将自动创建一个标准的多文档应用程序，如图 9-11 所示。

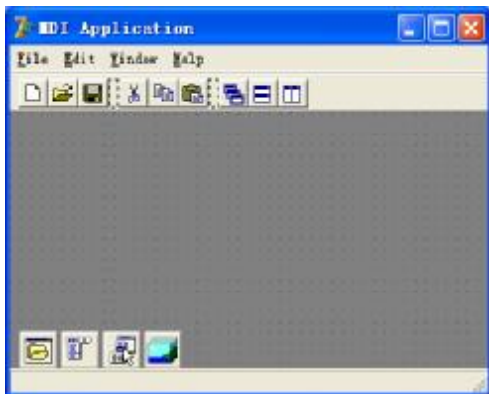


图 9-11 使用模板创建的 MDI 应用程序

创建的 MDI 应用程序是一个模板，它具有比较完善的功能。还可以向其中增加组件，修改程序代码来实现自己想要实现的功能。

3. 用户从头开始创建 MDI 应用程序

(1) 创建主窗体

一个 MDI 应用程序有一个主窗体，用来作为其他窗体的父窗口。要把某窗体设置为主窗体，需把它的 `FormStyle` 属性值设置为“`fsMDIForm`”。

(2) 创建子窗体

一个 MDI 应用程序可能有一个或多个不同风格的子窗口，每一种子窗口均需创建一个子窗体。要创建子窗体，首先应把窗体添加到应用程序中，然后把它的 `FormStyle` 属性值设置为“`fsMDIChild`”。

(3) 创建应用程序菜单和菜单融合

一般的 MDI 应用程序中，不但父窗口有菜单，子窗口也可以有菜单。如果子窗口有菜单，则程序运行时，当子窗口获得焦点时，子窗口的菜单项将与父窗口菜单融合。父窗口菜单与子窗口菜单的创建方法与普通窗体的菜单创建方法基本相同。菜单融合是指在程序运行过程中，子窗口菜单与父窗口菜单的相互作用。如当子窗口获得焦点时，子窗口的菜单或插入主窗口的菜单中，或替换部分或全部父窗口菜单。

为进行菜单融合，需设置窗体的 `Menu` 属性和菜单项的 `GroupIndex` 属性。

Menu 属性定义窗体的活动菜单，而菜单融合只针对活动菜单进行。如果窗体有多个菜单组件，运行时可通过以下代码选择当前使用的菜单组件：

```
Form1.Menu:=菜单组件名;
```

GroupIndex 属性决定出现在菜单条中各菜单项的位置。在菜单融合中，GroupIndex 属性将决定子菜单是插入还是替换主窗口菜单条中的菜单。GroupIndex 属性的默认值是 0，其取值及含义如下。

- ❶ 数值越小，菜单的位置越靠左。如 GroupIndex 为 0 的菜单将出现在菜单条的最左端。随着 GroupIndex 数值的增大，菜单项依次向右排列。
- ❷ 若需要替换主菜单中的某一菜单项，则将子菜单相应菜单项的 GroupIndex 设为与之相等的值。这条规则适合一个或多个菜单项。如主菜单中的“Temp”菜单的 GroupIndex 属性的值为 N。将子菜单的一个或多个菜单项的 GroupIndex 的值设为 N，则在运行时，这些菜单项将替换主菜单的“Temp”菜单。
- ❸ 将同一窗体的多个菜单项的 GroupIndex 设为相同值，原有的排列顺序在菜单融合时将保持不变。
- ❹ 若要在菜单融合时插入菜单项，则需要主菜单中预留 GroupIndex 数值“位置”。如主菜单的两菜单项数值为 0、2，则子菜单中 GroupIndex 数值为 1 的菜单在融合时将插入其中。

(4) 运行时子窗体的创建和关闭

通常在设计时要创建子窗体的模板，在程序执行时通过子窗体的 Create 方法创建子窗体。

要关闭子窗体，可使用子窗体的 Close 方法。此时将产生子窗体的 FormClose 事件，在该事件中应给 Action 参数赋值 caFree，如以下的 FormClose 事件程序：

```
procedure TForm1.FormClose(Sender: TObject;var Action: TcloseAction);  
begin  
    action:=cafree;  
end;
```

在窗体的 FormClose 事件中，Action 参数的取值及其含义如表 9-3 所示。

表 9-3 Action 参数值及其含义

参 数 值	含 义
caNone	窗体不允许关闭，因此什么都没有发生
caHide	窗体没有关闭，只是隐藏起来，应用程序还可以访问该窗体
caFree	关闭窗体并释放窗体所占用的内存
caMinimize	窗体被最小化，而不是关闭

在 MDI 子窗体的 FormClose 事件中如果不给 Action 参数赋值，系统默认把 caMinimize 赋值了 Action 参数。

(5) 与 MDI 应用程序有关的一些属性或方法

- ❶ Mdicildcount 属性：该属性是一个运行属性，用来记录当前共有多少个 MDI 子窗体。

- **Mdichildren** 数组属性：该属性用来引用 MDI 子窗体，它的每一个元素对应一个 MDI 子窗体，如以下程序段是把所有的子窗体给关闭：

```
for i:= mdichildcount-1 Downto 0 do  
    mdichildren[i].close;
```

- **Tile** 方法：该方法的作用是以平铺方式排列所有的子窗体。
- **Cascade** 方法：该方法的作用是以层叠方式排列所有的子窗体。

9.1.4 变量的作用域

能够使用变量的程序段称为变量的作用域。在单元的不同地方定义的变量其作用域是不同的，如果变量在一个过程或函数中定义，则该变量只能在该过程或函数中使用，这样的变量称为局部变量；如果变量在 **implementation** 部分中定义，则该变量能够被本单元内的所有过程或函数使用，可用来在本单元的各个过程或函数之间传递数据，这样的变量称为单元级（或模块级、或窗体级）变量；如果变量单元的 **interface** 部分中定义，则该变量能够被引用该单元的其他窗体和单元中的过程或函数访问，这类变量可以在单元或窗体之间传递数据，这样的变量称为全局变量。

【例 9-4】 编写一个七位号码体彩摇奖程序，程序中有两个窗体，分别如图 9-12 和图 9-13 所示。程序运行时首先出现图 9-12 所示的摇奖窗体，在该窗体上单击【摇奖】按钮，将不停地产生随机数并填充在七个文本框中，如图 9-14 所示。单击【停止】按钮将停止摇奖，此时在文本框中显示的数就是获奖号码。单击【公布】按钮将公布获奖号码，如图 9-15 所示。在该窗体上单击【退出】按钮将退出应用程序。

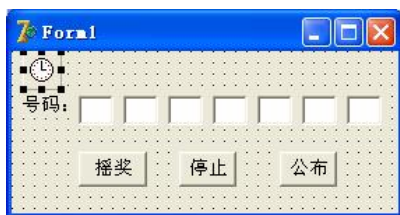


图 9-12 摇奖窗体设计界面



图 9-13 公布摇奖号码窗体设计界面



图 9-14 摇奖窗体运行界面

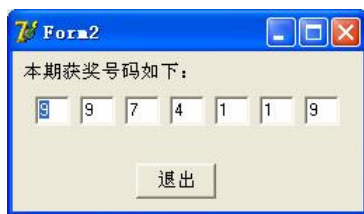


图 9-15 公布摇奖号码窗体运行界面

【实现分析】

可用七个整型变量存放获奖号码的七位数，这七位数是在 Form1 窗体上摇得的，并且可以在 Form2 窗体中公布，因此这七个变量应定义成全局变量。为实现摇奖功能，可使用一个

Timer 组件, 在该组件的 Timer 事件中产生七个 0~9 的随机数, 并显示在编辑框组件中。单击【摇奖】按钮, 把 Timer 组件的 Enabled 属性设置为 True, 开始摇奖, 单击【停止】按钮, 把 Timer 组件的 Enabled 属性设置为 False, 此时七个变量里的值就是获奖号码。

【界面设计】

把 Timer1 组件的 Enabled 属性值设置为 False, Interval 属性值设置为 150, 其他组件的属性设置请参照图 9-12 和图 9-13 进行。

【程序代码】

Form1 窗体的主要程序代码如下:

```
var
    Form1: TForm1;
var
    a1,a2,a3,a4,a5,a6,a7:integer;           //存放摇出来的号码
implementation
uses Unit2;
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
    Randomize;                             //随机数初始化
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    a1:=random(10);  a2:=random(10);
    a3:=random(10);  a4:=random(10);
    a5:=random(10);  a6:=random(10);
    a7:=random(10);           //以上是产生 7 个 0~10 之间的随机数
    Edit1.Text :=inttostr(a1);  edit2.Text :=inttostr(a2);
    Edit3.Text :=inttostr(a3);  edit4.Text :=inttostr(a4);
    Edit5.Text :=inttostr(a5);  edit6.Text :=inttostr(a6);
    Edit7.Text :=inttostr(a7);           //以上是把产生的随机数显示在相应的 Edit 组件中
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Timer1.Enabled :=True;               //开始摇奖
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    Timer1.Enabled :=False;              //停止摇奖
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    Form2.show;                          //显示公布摇奖号界面
end;
```

Form2 窗体的主要程序代码如下:

```
procedure TForm2.FormShow(Sender: TObject);
begin
    Edit1.Text :=inttostr(a1);  edit2.Text :=inttostr(a2);
    Edit3.Text :=inttostr(a3);  edit4.Text :=inttostr(a4);
    Edit5.Text :=inttostr(a5);  edit6.Text :=inttostr(a6);
    Edit7.Text :=inttostr(a7);           //以上语句是公布获奖号码
end;
procedure TForm2.Button1Click(Sender: TObject);
begin
    Form2.Close ;
    Form1.Close ;
end;
```

9.2 典型实例

【实例题目】：记事本程序

编写一个记事本应用程序，该程序能够实现文本文件的打开、修改、保存等文件操作，同时该程序还能够实现剪切、复制、粘贴等剪贴板操作。记事本程序的主设计界面如图 9-16 所示，主菜单如图 9-17 所示。



图 9-16 记事本运行界面



图 9-17 主菜单

【实现方法】

(1) 记事本程序是一个单文档应用程序，在任何时刻只能打开一个文件，因此无论是【打开】还是【新建】文件时，原先打开的文件必须先关闭。

(2) 本例使用 TRichEdit 组件来显示和修改文本文件。当执行【打开】操作的时候，可通过一个打开对话框选择一个文本文件，然后调用 TRichEdit 组件的 Lines.LoadFromFile 方法把相应文件的内容显示在 TRichEdit 组件中。执行文件的【保存】或【另存为】操作时，可调用 TRichEdit 组件的 Lines.SaveToFile 方法把内容存放到指定的文件中。

(3) 判断文件是否被修改。无论是在新建、打开文件时，还是在关闭文件时，均需判断当前文件是否被修改，如果被修改，应提示是否保存当前文件。判断文件是否被修改可使用 RichEdit 组件的 Modified 属性，属性值为 True 表示当前打开的文件被修改，否则没有被修改。

(4) 剪贴板操作的实现。剪切操作可通过调用 TRichEdit 组件的 CutToClipboard 方法来

实现，复制操作可通过调用 TRichEdit 组件的 CopyToClipboard 方法来实现，粘贴操作可通过调用 TRichEdit 组件的 PasteFromClipboard 方法来实现。

(5) 删除操作的实现。可通过把它的 Seltext 属性值设置为空字符串(“”)来实现。

(6) 取消操作的实现。可调用 Windows API 函数 SendMessage 传送 em_undo 来实现，em_undo 来自 Inprise 公司的源代码 REdit.pas，故使用它必须把 Redit 包含在 Uses 子句中。

【界面设计】

本例组件属性设置及组件作用如表 9-4 所示。

表 9-4 窗体及组件的属性设置及组件作用

对 象 名	属 性 名	设 置 值	对 象 作 用
Form1	Caption	'记事本'	主窗体
	Name	'MainForm'	
	Position	poScreenCenter	
MainMenu1			系统菜单（设置参见图 9-17）
RichEdit1	Align	alClient	显示和修改文件内容
OpenDialog1	Filter	'文本文件*.txt'	执行打开操作时弹出打开对话框
SaveDialog1	Filter DefaultExt	'文本文件*.txt' *.txt'	执行“另存放”操作时弹出另存为对话框
FontDialog1			执行“字体”设置操作时弹出字体对话框
PrintDialog1			执行“打印”操作时弹出打印对话框

【程序代码】

本例主要程序代码如下：

```
var
    MainForm: TMainForm;
    CurFname:string;           //该变量用来存放记事本当前打开文件的文件名
implementation
uses main;
{$R *.dfm}
    //该过程用来把打开或新建的文件的文件名保存到 CurName 变量中，同时在窗体的标题
    //栏显示文件名
procedure setFileName(const filename:string);
begin
    CurFname:=filename;
    MainForm.Caption:=format('%s-%s',[extractfilename(filename),application.title]);
end;
    //该过程用来检查文件是否修改，如果修改则提示是否保存
procedure CheckFileModi;
var    Resp:integer;
begin
    if not MainForm.RichEdit1.Modified then exit;
    Resp:=messagedlg(format('将修改的文件存入文件%s?',
        [CurFname]),mtConfirmation,mbYesNoCancel,0);
```

```
case Resp of
    idyes:mainform.RichEdit1.Lines.SaveToFile(CurFname);
    idno;;
    idcancel:abort;
end;
end;
// 该过程用来把参数 filename 指定的文件打开, 关设置文件名
procedure performfileopen(const filename:string);
begin
    mainform.RichEdit1.Lines.LoadFromFile(filename);
    setfilename(filename);
    mainform.RichEdit1.SetFocus;
end;
procedure TmainForm.N7Click(Sender: TObject);      // 【另存为】菜单项
begin
    if savedialog1.Execute then
    begin
        if fileexists(savedialog1.FileName) then
        if messagedlg(format(' 是否覆盖文件%s?', [savedialog1.FileName]),mtconfirmation,
            mbyesnocancel,0)<>idyes then exit;
        mainform.richedit1.Lines.SaveToFile(savedialog1.FileName);
        setfilename(savedialog1.FileName);
        mainform.richedit1.Modified:=false;
    end;
end;
procedure TmainForm.N8Click(Sender: TObject);      // 【打印】菜单项
begin
    if printdialog1.Execute then
        mainform.RichEdit1.Print(CurFname);
end;
procedure TmainForm.N10Click(Sender: TObject);     // 【取消】菜单项
begin
    with mainform.RichEdit1 do
        if handleallocated then sendmessage(handle,em_undo,0,0);
end;
procedure TmainForm.N11Click(Sender: TObject);     // 【复制】菜单项
begin
    mainform.RichEdit1.CopyToClipboard;
end;
procedure TmainForm.N12Click(Sender: TObject);     // 【粘贴】菜单项
begin
    mainform.RichEdit1.PasteFromClipboard;
end;
procedure TmainForm.N13Click(Sender: TObject);     // 【删除】菜单项
begin
    mainform.RichEdit1.SelText :='';
```

```
end;
procedure TmainForm.N16Click(Sender: TObject);      // 【剪切】菜单项
begin
    mainform.RichEdit1.CutToClipboard;
end;
procedure TmainForm.N14Click(Sender: TObject);      // 【字体】菜单项
begin
    if mainform.FontDialog1.Execute then
        mainform.RichEdit1.Font:=fontdialog1.Font;
end;
procedure TmainForm.N4Click(Sender: TObject);      // 【新建】菜单项
begin
    CheckFileModi;
    setfilename('未命名');
    mainform.richedit1.Lines.Clear;
    mainform.richedit1.Modified:=false;
end;
procedure TmainForm.N5Click(Sender: TObject);      // 【打开】菜单项
begin
    CheckFileModi;
    if opendialog1.Execute then
        begin
            performfileopen(opendialog1.FileName);
            mainform.richedit1.readonly:=ofreadonly in opendialog1.options;
        end;
end;
procedure TmainForm.N9Click(Sender: TObject);      // 【退出】菜单项
begin
    close;
end;
```

9.3 上机练习

【练习题目】：图片文件查看器

编写一个能够同时查看多张图片文件的图片文件浏览器，该应用程序是一个 MDI 应用程序，由两个窗体组成，一个名为 MDIfileVIEWER 的主窗体，一个名为 pictureViewer 的子窗体。MDIfileVIEWER 主窗体的设计界面如图 9-18 所示，该窗体有一个名为 OpenFileDialog 的打开对话框组件和一个名为 MainMenu1 的菜单组件，菜单的设计情况如图 9-19 所示。pictureViewer 窗体的设计界面如图 9-20 所示，该窗体上有一个名为 Image1 的图像框组件用来显示打开的图片文件，一个名为 MainMenu1 的菜单组件，菜单的设计情况如图 9-21 所示。



图 9-18 MDI 主窗体设计界面



图 9-19 MainMenu1 菜单设计情况



图 9-20 pictureViewer 子窗体设计界面



图 9-21 MainMenu1 菜单设计情况

程序执行时的初始界面如图 9-22 所示, 打开了若干个图片文件后的运行界面如图 9-23 所示。从该运行界面可以看到子窗口的菜单已经融合到了主窗口菜单中。

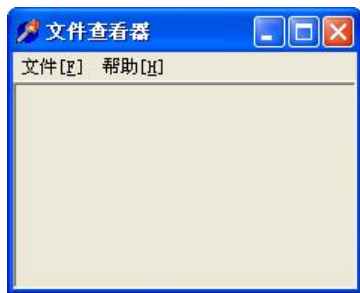


图 9-22 程序运行初始界面



图 9-23 程序运行界面

【要点提示】

(1) 菜单融合。当子窗体获得焦点时, 子窗体的菜单将和主窗体的菜单融合在一起。由图 9-23 可知, 子菜单的【文件】菜单替换了主菜单中的【文件】菜单, 子菜单中的【窗口】菜单插入在主菜单的【文件】和【帮助】菜单之间。为实现该功能, 本题可这样设置各菜单的 GroupIndex 属性值: 主窗体中的【文件】菜单的 GroupIndex 值为 0; 主窗体中的【帮助】菜单的 GroupIndex 值为 2; 子窗体中的【文件】菜单的 GroupIndex 值为 0; 子窗体中的【窗口】菜单的 GroupIndex 值为 1。

(2) 文件的打开。当执行文件的打开操作时, 通过一个打开对话框选择一个图片文件。如果选择的是图片文件, 则通过程序创建一个子窗体的实例, 并把图片文件显示在子窗体中。

的图像框控件中。需注意的是,通过 **Create** 方法创建的窗体,一开始时是不可见的,必须把它的 **Visible** 属性设置为 **True**,窗体才可见。

(3) 关闭所有子窗体。MDI 窗体有一个属性 **mdichildcount**,它的值代表着子窗体的数量,有一个数组属性 **mdichildren**,通过它可以访问每一个子窗体,通过一个循环执行每一个窗体的 **Close** 方法即可。

(4) 子窗体的卸载。应在窗体的 **FormClose** 事件中执行下面语句使窗体卸载:

```
action:=cafree;
```

如果没有该语句,则子窗体被最小化。

【参考代码】

主窗体 (MDIfileVIEWER) 的程序代码如下:

```
implementation
  uses pictureviewer;           //引用子窗口所在的单元文件
{$R *.DFM}
procedure tmdifileviewer.closeallchildren;   //该过程用来关闭所有子窗体
var
  i:integer;
begin
  for i:= mdichildcount-1 Downto 0 do      // 循环遍历所有的子窗体
    mdichildren[i].close;                  //关闭子窗体
  end;
procedure TMDIfileVIEWER.Exit1Click(Sender: TObject);    //【退出】菜单项
begin
  close; //关闭主窗体
end;
procedure TMDIfileVIEWER.Open1Click(Sender: TObject);    //【打开】菜单项
var
  PicViewer:tpictureviewer;           //该变量用来保存程序执行时生成的子窗体
  fileext:string[4];
begin
  if Fileopendialog.execute then
  begin
    fileext:=extractfilext(fileopendialog.filename);      //获得扩展名
    if (fileext='.BMP') OR (fileext='.JPG') then           //如果是图片文件
    begin
      PicViewer:=tpictureviewer.create(self);              //创建一个子窗口
      PicViewer.open(fileopendialog.filename);              //打开文件
      Picviewer.visible:=true;                               //使创建的窗体可见
      PicViewer.setfocus;                                   //把焦点设置在新创建的窗体上
    end
  else
    ShowMessage('不合法的图片文件扩展名');
  end;
end;
```

```
end;
```

查看图片的子窗体（pictureViewer）的程序代码如下：

```
implementation
uses viewmain;
{$R *.DFM}

procedure Tpictureviewer.open(const afilename:string);      // 【打开】方法
begin
    filename:=afilename;
    image1.picture.loadfromfile(filename);                 //装载图片
    caption:=filename;
end;

procedure TpictureViewer.Open1Click(Sender: TObject);      // 【打开】菜单项
begin
    mdifileviewer.open1click(sender);                      //调用主窗体的打开菜单项的事件过程
end;

procedure TpictureViewer.Close1Click(Sender: TObject);      // 【关闭】菜单项
begin
    close;
end;

procedure TpictureViewer.Exit1Click(Sender: TObject);      // 【退出】菜单项
begin
    mdifileviewer.exit1click(sender);
end;

procedure TpictureViewer.Tile1Click(Sender: TObject);      // 【平铺】菜单项
begin
    Mdifileviewer.tile;
end;

procedure TpictureViewer.Cascade1Click(Sender: TObject);    // 【层叠】菜单项
begin
    mdifileviewer.cascade;
end;

procedure TpictureViewer.All1Click(Sender: TObject);        // 【关闭全部】菜单项
begin
    mdifileviewer.closeallchildren;
end;

procedure TpictureViewer.FormClose(Sender: TObject;
    var Action: TCloseAction);
begin
    action:=cfree;                                          //使窗体卸载
end;
```

课后考场

一、选择题（20分，每题5分）

- MDI 应用程序的概念是_____。
A. 一个时刻只能打开一个窗口
B. 只要有多个窗口的应用程序就是 MDI 应用程序
C. MDI 应用程序由一个 MDI 主窗体和若干个子窗体组成，子窗体无法移出主窗体
D. MDI 应用程序就是可以通过程序创建窗体的应用程序
- 在单元的 **Interface** 部分定义的变量，其作用域是_____。
A. 作用域不明，由用户再根据使用情况决定
B. 全局变量，能被本单元中的过程或函数以及包含该单元的单元中的过程或函数使用
C. 模块（窗体或单元）级变量，只能被本单元中的过程或函数使用
D. 局部变量，只能被特定的过程或函数使用
- 关闭 MDI 子窗体可使用 **Close** 方法，此时将产生子窗体的 **FormClose** 事件，在该事件中应给 **Action** 参数赋值_____。
A. **caFree** B. **caHide** C. **caMinimize** D. **caNone**
- 如果要把某个窗体设置为 MDI 主窗体，应_____。
A. 把它的 **FormStyle** 属性值设置为 “**fsMDIForm**”
B. 把它的 **FormStyle** 属性值设置为 “**fsMDIChild**”
C. 把它的 **FormStyle** 属性值设置为 “**fsNormal**”
D. 把它的 **FormStyle** 属性值设置为 “**fsStayOnTop**”

二、填空题（40分，每空5分）

- 为实现 MDI 主窗体的菜单与 MDI 子窗体的菜单融合，应设置菜单的_____属性。
- 为了通过模板创建一个 SDI 窗体，首先应执行【**File**】→【**New**】→【_____】命令。
- MDI 子窗体的数量可以由 MDI 主窗体的_____属性来得到。
- 要访问各个 MDI 子窗体，可使用 MDI 主窗体的_____属性，该属性是一个数组属性。
- 要把所有的 MDI 子窗体平铺，应执行 MDI 主窗体的_____方法。
- 某一个变量，如果希望它能被本单元中的所有过程和函数使用，但不能被其他单元中的过程和函数使用，应在单元的_____部分定义该变量。
- 在过程或函数中定义的变量是_____变量。

三、程序设计题（40分）

编写一个文本文件与图片文件的查看程序，程序运行时能够打开多个文本文件和图片文件，还可以通过菜单对子窗口进行层叠、平铺等操作，还可以关闭所有窗口。程序的运行界面如图 9-24 所示。（提示：在本章上机练习一的基础上加上查看文本文件内容的功能，故应再设计一个查看文本文件的子窗体，子窗体的设计界面如图 9-25 所示，该子窗体的菜单与图

片查看窗口的菜单完全一致。)



图 9-24 程序运行界面



图 9-25 【文本文件查看窗口】设计界面

第 10 章 DLL 应用编程

本章要点

- ▮ DLL 的概念
 - ▮ 动态链接库的编写
 - ▮ 动态链接库的隐式调用和显示调用
 - ▮ 利用动态链接库实现窗体重用的方法
-

10.1 理论知识

动态链接库 (Dynamic Link Library, DLL) 是实现 Windows 应用程序代码重用和共享的重要手段, 它是从 C 语言函数库和 Pascal 语言的库单元的概念发展起来的。在 C 和 Pascal 语言中, 很多标准函数可以放在一个函数库中或一个库单元中, 在用户程序中可以调用这些库单元或函数库中的函数。在程序编译的时候, 由编译器把所调用的函数添加到可执行文件中 (即每个调用均会产生一个函数代码的拷贝)。使用函数库或库单元等方法已不能适用 Windows 多任务环境下应用程序, 因为使用这种方法每个应用程序都必须拥有完成屏幕输出、消息处理、内存管理、对话框操作都相同功能的函数, 从而使应用程序非常庞大, 为解决问题, 必须能使几个应用程序能共享函数的单一拷贝, DLL 也就应时而生了。

10.1.1 DLL 概述

1. DLL 的概念

DLL 是一个可以执行的并可以被多个 Windows 应用程序共享的程序模块 (Module), 模块中包含了一些可以被多个 Windows 应用程序或 DLL 共享的代码、数据和资源。动态链接库不用重复编译或链接, 一旦装入内存, DLL 中的函数就可以被系统中的任何正在运行的应用程序所使用, 而不必产生函数的多个拷贝。DLL 主要的用途是使应用程序可以在运行时刻载入其中的代码, 而不是在编译时连接到应用程序中。

DLL 文件的扩展名一般是 dll, 也有可能是 drv, sys 和 fon (它们分别对应的是设备驱动文件、系统文件和字体资源文件)。它和可执行文件 (exe) 非常类似, 区别在于, DLL 文件中虽然包含了可执行代码却不能单独执行, 只能由 Windows 应用程序直接或间接调用。

DLL 类似于一般的运行函数库, 但它与一般的运行库函数又有区别, 区别主要在于, DLL 与一般的函数库与应用程序链接的时机不同。DLL 是在应用程序运行期间被链接进来的, 而一般函数库是在应用程序使用链接器 (Linker) 链接文件时被链接进来的。这两种链接方式分别称为动态链接与静态链接。下面介绍这两种链接方式的概念。

2. 静态链接和动态链接

(1) 静态链接

传统的库函数是通过静态链接链接到应用程序中的。这些库函数可能来自程序设计语言提供的标准库，也可能是由操作系统提供的 API。由应用程序源文件产生可执行文件，一般要经历两个过程：一是将源程序编译成目标模块，在此阶段源代码中，凡调用到库函数的地方都被标识为外部对象的引用；二是把目标模块链接成 EXE 文件，该阶段将所有要用到的函数均拷贝一份，插入到应用程序的目标模块文件中，从而生成可执行文件。也就是说静态链接是将应用程序调用的库函数拷贝一份嵌入到应用程序的可执行文件中去。

静态链接可用图 10-1 来描述（椭圆表示操作）。

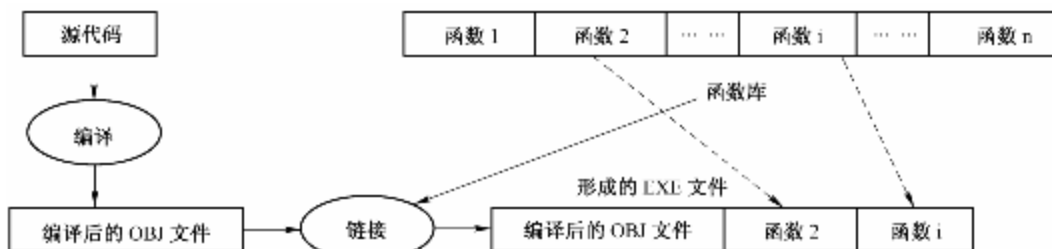


图 10-1 静态链接示意图

可见，使用静态链接每调用一个函数均会在可执行文件的目标代码中出现一个该函数代码拷贝，当多个应用程序调用相同的函数时，将会出现一个函数的多个拷贝，其缺陷是很明显的。

(2) 动态链接

动态链接是指在把应用程序的目标代码链接成 EXE 文件时，没有将函数库中的函数拷贝到应用程序的可执行文件中，而是在程序运行时动态地加载所需的函数。采用动态链接方式的库文件就是动态链接库（DLL）。尽管链接器并不把动态链接的函数拷贝到可执行文件中，但是它仍然要清楚这些函数在什么地方以及怎样调用它们，为此需要引入库（Import Library）来帮助链接器使用 DLL，引入库中包含了 DLL 中函数的重定位信息。当应用程序使用了某个 DLL 中的一个函数时，链接器并不拷贝代码，而是从引入库中拷贝重定位信息，这些信息指示了运行期间在 DLL 的什么位置寻找所需的函数代码。在应用程序执行期间，这些重定位信息创建了一个正在执行的应用程序和在内存在中的 DLL 之间的“动态链接”。图 10-2 为动态链接的示意图。

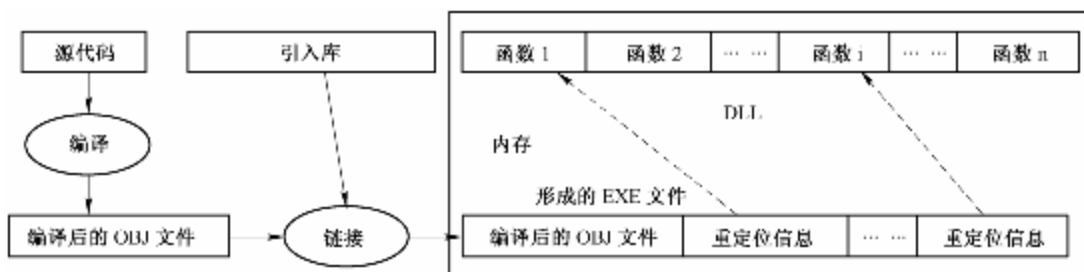


图 10-2 动态链接示意图

可见,在动态链接时,应用程序和所需的 DLL 均应加载进入内存,根据应用程序中的重定位信息找到 DLL 中相应的函数,从而可以执行该函数功能以完成函数的调用。

当多个应用程序调用同一个 DLL 文件中的同一个函数时,不会生成该函数的多个拷贝,DLL 文件也只在内存中存在一份,因此使用 DLL 不但便于应用程序之间共享代码,而且有利于节省内存。

3. 使用 DLL 的优点

(1) 应用范围广。DLL 中不仅可以包含可执行代码,还可以包括数据和各种资源等,这扩大了库文件的使用范围。有些设备驱动程序也是由动态链接库实现的(扩展名一般是 `drv`)。

(2) 便于开发大型软件。一个大型系统,如果用一个可执行文件完成,程序将很庞大,而且其中可能有许多重复的功能。如果将程序分成一系列的主程序和 DLL,可以减少开发的工作量。并且,由于每个模块减小了,访问的速度也提高了。

(3) 便于对系统进行升级。将一些功能模块做成 DLL 后,如果需要对系统进行升级,只要将个别 DLL 进行升级,然后用新的 DLL 文件覆盖掉旧的 DLL 文件就可以了,而不需要将整个系统进行重新编译和链接。

(4) 隐藏细节。例如,有一件工作的完成方法有许多,可以将这些方法利用 DLL 实现,当以后新增加了方法后,将新方法也用 DLL 实现,然后只要对原来的工程文件做少量的修改就可以了。这样,就隐藏了 DLL 的实现细节,减少了使用该 DLL 的客户程序的实现复杂性,从而增加了重用的可能性。

(5) 独立于程序设计语言。DLL 是符合一定标准的使用不同语言进行编写的函数库。例如:在 Delphi 环境中开发的 DLL 库可以在 Visual C++ 环境中方便地使用,同时,在 Delphi 环境中也可以方便地使用在 Visual C++ 中开发的 DLL。

10.1.2 DLL 编写

根据 DLL 完成的功能,可把 DLL 分成三类:完成一般功能的 DLL、用于数据交换的 DLL 和用于窗体重用的 DLL。下面通过一个实际的例子来讲解完成一般功能的 DLL 的编写方法。

【例 10-1】 创建一个 DLL,该 DLL 中包含两个函数,分别用于求三个数的最大值和最小值。

编写该 DLL 的步骤如下。

(1) 建立 DLL 程序框架

在 Delphi 7 集成环境中,执行【File】→【New】→【Other】命令,将会出现【New Items】对话框,选择【DLL Wizard】图标,如图 10-3 所示。然后单击【OK】按钮,将会自动创建一个 DLL 工程,并在代码编辑器中打开了该工程,默认的 DLL 工程文件名为“Project1.dpr”或“Project2.dpr”等。

新建的 DLL 项目文件内容一般如下:

```
library Project2;
{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
```


functions that pass strings as parameters or function results. This applies to all strings passed to and from your DLL--even those that are nested in records and classes. ShareMem is the interface unit to the BORLNDMM.DLL shared memory manager, which must be deployed along with your DLL. To avoid using BORLNDMM.DLL, pass string information using PChar or ShortString parameters. }

```
uses
  SysUtils, Classes;
{$R *.res}
begin
end.
```

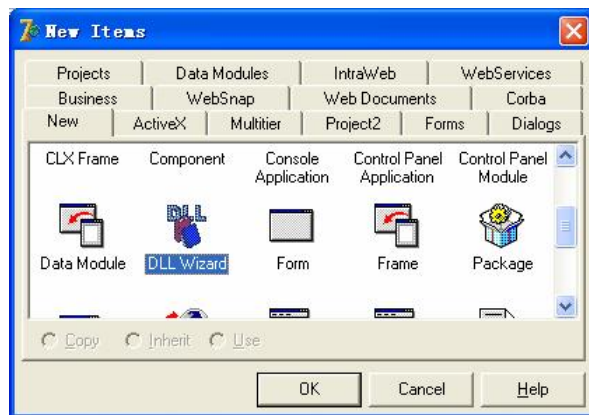


图 10-3 【New Items】对话框

注意：

■ 注释的中文含义是：在 DLL 中，如果导出过程或函数的参数为字符串或动态数组，或者函数的返回值为字符串或动态数组，则在 uses 语句中应包含 ShareMem 单元，且要将 ShareMem 单元放在最前。ShareMem 是共享的内存管理器 Borlandmm.dll 的接口单元，Borlandmm.dll 必须与 DLL 一起发布。另外一种解决办法就是利用 PChar 或 ShortString 类型来传递字符串信息，可以不用在 uses 语句中包含 ShareMem 单元。

■ 在 begin 和 end 之间可以编写 DLL 的初始代码。

DLL 项目创建后，应把它保存起来，本例保存到文件夹“D:\DelphiApp\10\A\ A_10_1”下，取名为“Exam_10_1”，会发现 Library 后的库名随之改变。

(2) 建立过程和函数

为该 DLL 项目增加如下两个函数：

```
Function Max(x,y,z:Integer):Integer;stdcall;           //求三个数的最大值函数
var
  t:integer;
begin
  if (x<y) then
    t:=y
```

```

else
    t:=x;
if (t<z) then
    t:=z;
max:=t;
end;
Function Min(x,y,z:Integer):Integer;stdcall;           //求三个数的最小值函数
var
    t:integer;
begin
    .....           //此处省略求三个数的最小值，与求三个数的最大值的方法基本相同
end;

```

注意：定义函数时使用的“stdcall”是一种调用约定，如果程序员希望自己的 DLL 库函数能够被其他程序设计语言的程序调用，应使用 stdcall 调用约定。默认的调用约定为 register，该约定是最快的参数传递约定，但其他程序设计语言可能不支持该调用约定。主要的调用约定关键字如表 10-1 所示。

表 10-1 调用约定关键字

关 键 字	参数传递顺序	使 用 场 合
register	自左向右	默认，使用寄存器的参数传递方式
pascal	自左向右	主要用于向前兼容的目的
cdecl	自右向左	多用于使用 C 或 C++ 语言编写的函数
stdcall	自右向左	主要用于调用 Windows API 函数
Safecall	自右向左	主要用于调用 Windows API 函数和双重接口

(3) 用 exports 语句声明供其他应用程序调用的函数和过程名

函数或过程定义好后，为了能够被其他应用程序调用，还必须用 Exports 子句把函数名列出。本例使用的 Exports 语句如下：

```

exports
    Max,Min;

```

Exports 语句的格式与功能如下：

```

exports
    entry1,entry2,...,entryn;

```

其中 entry1、entry2 等称为条目，一般包括如下内容。

- ① 函数、过程或变量名。
- ② 参数列表。如果声明的函数或过程被重载，则后面还需要跟相应的参数列表。
- ③ name 指示符。用 name 指示符可给函数或过程再定义一个名称，作为该过程或函数的输出名，如：

```
exports
    Max name 'MaxInt';
```

则其他应用程序将用 `MaxInt` 调用名为 `Max` 的过程或函数。

④ `index` 编号。`index` 指示为过程或函数分配一个顺序号, 如果不使用 `index` 编号, 则 Delphi 7 编译器将按照顺序进行分配, 其数字范围是 1~32767, 使用 `index` 可以加速调用过程或函数。主要是为了向后兼容, 不提倡使用。

⑤ `resident` 指示符。使用 `resident`, 则当 DLL 载入时, 特定的输出信息始终保持在内存中, 这样, 当其他应用程序调用该过程时, 可以利用名字扫描 DLL 函数的入口以降低时间开销。该指示符主要是为了向后兼容, 不提倡使用。

注意:

! `exports` 子句可以出现在一个程序或库的声明部分, 也可以出现在单元文件的 `interface` 或 `implementation` 中的任意位置, 出现的次数也没有限制。但一般来说, 程序中很少使用 `exports` 语句。

! 当要输出重载的函数或过程时, 必须要 `exports` 子句中指明参数列表, 例如:

```
exports
    Max(x,y,z:integer) Name 'MaxInt';
    Max(x,y,z:Real) Name 'MaxReal';
```

(4) 生成 DLL 文件

执行【Project】→【Build Exam_10_1】命令(“Exam_10_1”为 DLL 项目文件名), 将自动生成名为“Exam_10_1.dll”的 DLL 文件。

10.1.3 加载 DLL 的方法

要调用 DLL 中的函数, 首先必须把 DLL 的文件映像映射到调用进程的地址空间中。有两种方法可以实现这一映射: 一种是在装入时动态链接 (Load-Time Dynamic Linking), 又称静态载入; 另一种是在运行时动态链接 (Run-Time Dynamic Linking), 又称动态载入。

1. 静态载入 DLL

这种方法是将 DLL 文件映像映射到调用进程地址空间的最简单的办法。在对应用程序进行编译时, 对每一个 DLL 函数的调用都会生成一个外部引用。为解析这些外部调用, 在链接时将使用一个 DLL 引入库 (.LIB) 文件, 该文件中含有 DLL 文件允许外部应用程序调用的函数列表, 其中包含了每一个 DLL 导出函数的符号名和可选的标识号, 但是并不含有实际的代码, 同时该文件中也包含了对应的 DLL 文件名(但不是完全的路径名), 导入库文件作为 DLL 的替代文件被编译到应用程序项目中。当链接器看到应用程序调用了 DLL 对应的 LIB 文件中列出的函数时, 就在生成的 EXE 文件中加入相关信息 (LIB 文件中导出符号或标识号), 指出包含所调用函数的 DLL 文件名。当应用程序运行时, 操作系统在装载应用程序时要查看 EXE 文件映像的内容, 并根据这些信息发现并加载 DLL 文件, 并通过符号名或标识号实现对 DLL 函数的动态链接。

静态载入 DLL 的前提是在编译之前已经明确知道要调用 DLL 中的哪几个函数, 编译时

在目标文件中只保留必要的链接信息，而不包含 DLL 函数的代码。在程序执行时，利用链接信息加载 DLL 函数代码并在内存中将其链接进入调用程序的执行空间，从而实现了代码的重用。

2. 动态载入 DLL

动态载入方式是指在编译之前并不知道将会调用哪些 DLL 函数，完全是在运行过程中根据需要决定应调用哪些函数，并用 LoadLibrary 函数加载动态链接库到内存，用 GetProcAddress 函数动态获得 DLL 函数的入口地址。当一个 DLL 文件用 LoadLibrary 显式加载后，在任何时刻均可以通过调用 FreeLibrary 函数显式地从内存中把它给卸载。

动态调用使用的 Windows API 函数主要有 3 个，分别是 LoadLibrary、GetProcAddress 和 FreeLibrary。下面分别介绍这三个函数的功能。

(1) LoadLibrary 函数

[格式]:

```
function LoadLibrary(LibFileName:PChar):THandle;
```

[功能]: 加载由参数 LibFileName 指定的 DLL 文件。

[说明]: 参数 LibFileName 指定了要装载的 DLL 文件名，如果 LibFileName 没有包含一个路径，系统将按照：当前目录、Windows 目录、Windows 系统目录、包含当前任务可执行文件的目录、列在 PATH 环境变量中的目录等顺序查找文件。如果函数操作成功，将返回装载 DLL 库模块的实例句柄，否则，将返回一个错误代码，错误代码的定义如表 10-2 所示。

表 10-2 LoadLibrary 返回错误代码的意义

错 误 代 码	意 义
0	系统内存不够，可执行文件被破坏或调用非法
2	文件没有被发现
3	路径没有被发现
5	企图动态链接一个任务错误或者有一个共享或网络保护错误
6	库需要为每个任务建立分离的数据段
8	没有足够的内存启动应用程序
10	Windows 版本不正确
11	可执行文件非法。或者不是 Windows 应用程序，或者在 .EXE 映像中有错误
12	应用程序为一个不同的操作系统设计（如 OS/2）
13	应用程序为 MS DOS 4.0 设计
14	可执行文件的类型不知道
15	试图装载一个实模式应用程序（为早期 Windows 版本设计）
16	试图装载包含可写的多个数据段的可执行文件的第二个实例
19	试图装载一个压缩的可执行文件。文件必须被解压后才能被装载
20	DLL 文件非法
21	应用程序需要 32 位扩展

假如在应用程序中用 LoadLibrary 函数装入某一个 DLL 前，其他应用程序已把该 DLL 装入内存中了，则系统将不再装入该 DLL 的另一个实例，而是使该 DLL 的“引用计数”加 1。

(2) GetProcAddress 函数

[格式]:

```
function GetProcAddress(Module:THandle;ProcName:PChar):TfarProc;
```

[功能]: 返回参数 Module 指定的模块中, 由参数 ProcName 指定的过程或函数的入口地址。

[说明]: 参数 Module 包含被调用函数的 DLL 句柄, 这个值由 LoadLibrary 返回, ProcName 是指向含有函数名的以 nil 结尾的字符串指针, 或者可以是函数的次序值, 但大多数情况下, 用函数名是一种更稳妥的选择。如果该函数执行成功, 则返回 DLL 中由参数 ProcName 指定的过程或函数的入口地址, 否则返回 nil。

(3) FreeLibrary 函数

[格式]:

```
procedure FreeLibrary(Module:THandle);
```

[说明]: 将由参数 Module 指定的 DLL 文件从内存中卸载 1 次。

[说明]: Module 为 DLL 库的句柄。这个值由 LoadLibrary 返回。由于 DLL 在内存中只装载一次, 因此调用 FreeLibrary 首先使 DLL 的引用计数减 1, 如果计数减为 0 则卸载该 DLL。

注意: 每调用一次 LoadLibrary 函数就应调用一次 FreeLibrary 函数, 以保证不会有多余的库模块在应用程序结束后仍留在内存中。

10.1.4 调用 DLL 中的过程和函数

在应用程序中, 调用 DLL 也有两种方法, 分别是隐式调用和显示调用。

1. 隐式调用

隐式调用又称静态调用或装载时调用, 对应于 DLL 的静态载入。要在应用程序中隐式调用某个动态链接库中的函数, 一般要用 extern 子句声明要调用的过程或函数及其所在的 DLL 文件名, 并在应用程序中直接调用用 extern 子句声明的过程。

注意:

- external 声明可以定义在任意单元的接口或实现部分。如果声明在实现部分, 它们只对声明的单元本地是可用的; 如果声明在接口部分, 则任何包括了声明单元的单元都可以访问它们。

- 如果使用 extern 子句声明过程, 则应用程序在运行时立即装载 DLL 并解析该过程或函数的地址。当 DLL 不再使用时, Windows 将负责卸载它们。

【例 10-2】 编写一个应用程序, 调用例 10-1 所建立的 DLL 中的 Max 和 Min 函数, 要求使用隐式调用。程序的设计界面如图 10-4 所示, 函数的运行界面如图 10-5 所示。程序运行时输入三个数到三个编辑框中, 然后单击【求最大值】按钮, 将会求出三个数的最大值并显示在第四个文本框中, 如果单击【求最小值】按钮, 将会求出三个数的最小值并显示在第四个文本框中。

【实现分析】

为隐式调用 DLL 中的函数, 需在应用程序中用 extern 子句对被调用函数进行声明, 然后在应用程序中就可以像调用普通函数一样调用 DLL 中的函数了。为了能够顺利地查找到函

数，可把例 10-1 产生的 Exam_10_1.DLL 文件复制到该程序所在的目录。

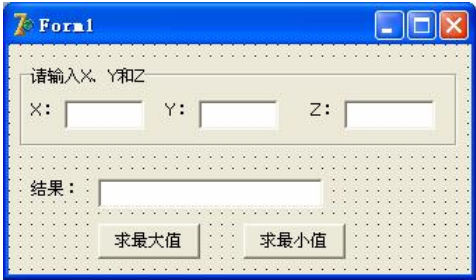


图 10-4 例 10-2 程序设计界面



图 10-5 例 10-2 程序运行界面

【界面设计】

本例的组件属性设置及组件功能如表 10-3 所示。

表 10-3 例 10-2 组件属性设置及组件功能

组 件 名	属 性 名	属 性 值	作 用
GroupBox1	Caption	'请输入 x、y 和 z'	
Label1	Caption	'x:'	
Edit1	Text	"	输入 x 的值
Label2	Caption	'y:'	
Edit2	Text	"	输入 y 的值
Label3	Caption	'z:'	
Edit3	Text	"	输入 z 的值
Label4	Caption	'结果:'	
Edit4	Text	"	显示求得的最大值或最小值
Button1	Caption	'求最大值'	单击它将求出三个数的最大值
Button2	Caption	'求最小值'	单击它将求出三个数的最小值

【程序代码】

主要程序代码如下：

```
var
  Form1: TForm1;
//下面用 External 子名声明 max 和 min 函数
Function Max(x,y,z:integer):Integer;stdcall;external 'EXAM_10_1.DLL';
Function Min(x,y,z:integer):Integer;stdcall;external 'EXAM_10_1.DLL';
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject); //求输入的三个数的最大值
var
  x,y,z,max_v:integer;
begin
  x:=strtoint(Edit1.Text ); y:=strtoint(edit2.Text );
```

```

z:=strtoint(edit3.Text );
max_v:=max(x,y,z);           //调用 max 函数求输入的三个数的最大值
Edit4.Text :=inttostr(max_v);
end;
procedure TForm1.Button2Click(Sender: TObject); //求三个数的最小值
var
    x,y,z,min_v:integer;
begin
    x:=strtoint(Edit1.Text ); y:=strtoint(edit2.Text );
    z:=strtoint(edit3.Text );
    min_v:=min(x,y,z);        //调用 min 函数求输入的三个数的最小值
    Edit4.Text :=inttostr(min_v);
end;

```

隐式调用 DLL 中的函数，程序实现比较简单，代码较少，但也有以下一些不足。

- (1) 如果加载的 DLL 文件不存在或者 DLL 中没有要引入的例程，程序将自动终止运行；
- (2) DLL 文件一旦加载就一直驻留在应用程序的地址空间，即使后面已不再使用它了。

2. 显式调用

使用显式调用（又称动态调用或运行时调用，对应于 DLL 的动态载入）的方法就可以克服隐式调用的两个缺点。显式调用 DLL 中的函数一般需经历以下步骤：

- (1) 定义一个与调用函数一致的函数类型；
- (2) 用 LoadLibrary 或 SafeLoadLibrary 函数动态载入 DLL；
- (3) 用 GetProcAddress 得到要调用的函数或过程的地址；
- (4) 利用定义的函数类型生成一个同类型的函数变量（本例为 Myfuncnt）；
- (5) 进行函数调用（本例调用语句为：Myfuncnt(x,y,z);）；
- (6) 利用 FreeLibrary 函数卸载 DLL。

【例 10-3】 编写一个应用程序，调用例 10-1 所建立的 DLL 中的 Max 和 Min 函数，要求使用显示调用。程序的设计界面和运行界面及程序功能同例 10-2。

【实现分析】

由于本例要调用的两个函数，它们的返回值和自变量的类型均是一样的，因此只需要定义一种函数类型。在程序运行时，单击不同的按钮将调用不同的函数，为了简化代码，可再编写一个函数（本例为 Calculate），其参数是要调用的函数名，其功能是显式调用以“函数参数”作为“函数名”的函数，并把调用结果作为函数的返回值。在按钮单击事件中首先形成函数名，然后调用 Calculate 函数来得到结果。

【界面设计】

本例的组件属性设置及功能如表 10-3 所示。

【程序代码】

主要程序代码如下：

```

implementation
type
    TintFunction=Function(x,y,z:integer):integer;stdcall; //定义函数类型

```

```

var
    x,y,z:integer;
{$R *.dfm}
Function Calculate(FunName:Pchar):integer;
var
    Dllname:string;           //存放 DLL 文件名
    Hinst:THandle;           //加载 DLL 文件的句柄
    ProcName:Pchar;          //要调用的函数或过程名
    Fpointer:TFarProc;        //函数指针
    MyFunct:TIntFunction;     //函数变量
begin
    getdir(0,dllname);
    dllname:=dllname+'\EXAM_10_1.dll';           //形成 DLL 文件名
    ProcName:=FunName;
    Hinst:=SafeLoadLibrary (Dllname);           //加载 DLL
    if Hinst>0 then                             //加载成功
        try
            Fpointer:=GetProcAddress(Hinst,ProcName); //得到要调用过程的地址
            if Fpointer<>Nil then
                begin
                    Myfunct:=TIntFunction(Fpointer); //形成函数名
                    Calculate:=Myfunct(x,y,z);       //调用相应函数
                end
            else
                showmessage('DLL Function not found.');
```

Finally

```

            FreeLibrary (Hinst);           //卸载 DLL
        end
    else
        ShowMessage(Dllname+' Library not Found');
```

end;

```

procedure TForm1.Button1Click(Sender: TObject);
var
    Funname:Pchar;           //函数名
    Res:integer;             //调用 DLL 的结果
begin
    Funname:='Max';
    x:=strtoint(Edit1.text);  y:=strtoint(Edit2.text);
    z:=strtoint(edit3.Text ); Res:=Calculate(FunName); //调用求最大值函数
    Edit4 .text:=IntToStr(Res) ;
end;
procedure TForm1.Button2Click(Sender: TObject);
var
```



```
Funname:Pchar;  
Res:Integer;  
begin  
    Funname:='Min';  
    x:=strtoint(Edit1.text); y:=strtoint(Edit2.text);  
    z:=strtoint(edit3.Text ); Res:=Calculate(FunName);           //调用求最小值函数  
    Edit4 .text:=IntTostr(Res) ;  
end;
```

10.1.5 在 DLL 中实现窗体重用

利用 Delphi 的 DLL 功能,不但能够实现过程和函数重用,而且还可以实现窗体重用。这样当用户编写了一个通用窗体或一个自己很满意的窗体,希望能在其他应用程序中多次使用的时候,就可以把窗体存放到 DLL 文件中,在需要的时候进行调用。同时把窗体编译在 DLL 中,还可以被其他语言(如 Visual C++、Visual Basic)编写的应用程序调用。

通常可以把一个或多个窗体编译在一个 DLL 中,以达到减少系统开销的目的。

利用 DLL 实现窗体重用一般需经历以下步骤:

- (1) 在 Delphi 的集成开发环境中,设计出需要重用的窗体;
- (2) 编写一个用于输出的函数或过程,在该函数或过程中,对设计的窗体进行创建使它实例化;
- (3) 如果要把多个窗体编译在一个 DLL 文件中,可重复步骤(1)和(2),直到所有需要的窗体建立完毕,所有对窗体进行实例化的函数或过程创建完毕;
- (4) 为适应生成 DLL 文件的需要,对工程文件进行相应修改;
- (5) 编译工程文件以生成 DLL 文件;
- (6) 在需要该窗体的其他应用程序中重用该窗体。

下面通过一个实例来讲解在 DLL 中实现窗体重用的方法。

【例 10-4】 在许多应用程序中都有在两个列表框中移动选项的窗体,请设计一个这样的可重用窗体,设计界面如图 10-6 所示。该窗体运行时,在左边的列表框中选中若干个选项后单击【>】按钮,将把选中的选项移动到右边的列表框中;在右边的列表框中选中若干个选项后单击【<】按钮,将把选中的选项移动到左边的列表框中;单击【>>】按钮将把左边的列表框中的所有选项移动到右边的列表框中;单击【<<】按钮将把右边的列表框中的所有选项移动到左边的列表框中。然后再编写一个应用程序用来调用该窗体。

实现本例功能的步骤如下。

- (1) 设计窗体。

新建一个应用程序,按图 10-6 所示给 Form1 添加组件对象。然后按表 10-4 所示设置组件对象属性。



图 10-6 设计界面

表 10-4 重用窗体的组件属性设置及组件作用

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'源列表框'	
Label2	Caption	'目标列表框'	
ListBox1	Items MultiSelect	'教授'副教授'讲师'助教'未定职' True	
ListBox2	Items	"	
Button1	Caption	'>'	把左边选中的选项移到右边
Button2	Caption	'>>'	把左边的全部选项移到右边
Button3	Caption	'<'	把右边选中的选项移到左边
Button4	Caption	'<<'	把右边的全部选项移到左边
Button5	Caption	'确定'	退出窗体，保存选择
Button6	Caption	'取消'	退出窗体，不保存选择

程序代码如下：

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i:integer;
begin
    For i:=ListBox1.Items.Count-1 Downto 0 do
        if Listbox1.Selected[i] then
            begin
                Listbox2.Items.Add(ListBox1.Items.Strings[i]) ;
                ListBox1.Items.delete(i);
            end;
    end;
procedure TForm1.Button3Click(Sender: TObject);
var
    i:integer;
begin
    .....//此处省略把 ListBox2 中的选中选项移动到 ListBox1 列表框中
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    ListBox2.Items.AddStrings(ListBox1.Items) ;ListBox1.Clear ;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    ListBox1.Items.AddStrings(ListBox2.Items) ; ListBox2.Clear ;
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
    modalResult:=mrOk ;
end;

```

```

procedure TForm1.Button6Click(Sender: TObject);
begin
    ModalResult:=mrCancel;
end;

```

(2) 编写输出函数或过程，实例化窗体

在窗体中再定义一个函数，其他程序可以通过调用该函数来获得对库的访问权。本例定义的函数名为 **ListMoveF**，其作用是创建窗体，并把窗体的执行结果——两个列表框中的选项数目作为参数返回。为了让其他语言也可以调用该函数，将其返回类型说明为 **Windows** 数据类型 **WordBool**，而不是 **Pascal** 语言特有的 **Boolean**。函数代码如下：

```

Function ListMoveF(Var  I1,I2:integer):WordBool;
begin
    Result:=False;
    Form1:=TForm1.Create(Application);
    try
        if Form1.ShowModal =mrOK then
            with form1 do
                begin
                    I1:=ListBox1.Items.Count ;
                    I2:=ListBox2.Items.Count ;
                    Result:=True;
                end;
            finally
                Form1.Free;
            end;
        end;
    end;
end;

```

函数定义好后，应在窗体单元的接口部分再对它进行说明，说明语句最后是关键字 **export**，它告诉编译器这个函数可以被 **DLL** 的客户程序调用。说明语句如下：

```
Function ListMoveF(Var  I1,I2:Integer):WordBool;Export;
```

(3) 修改工程文件，使之能生成 **DLL** 文件

窗体单元编制完成之后，可把应用程序项目转换为 **DLL**。首先执行【Project】→【View Source】命令打开 **.dpr** 项目文件，并在代码编辑器中做如下修改：

- ① 把 **program** 改为 **Library**;
 - ② 从 **uses** 指令中删除 **Forms**;
 - ③ 在 **{ \$R }** 资源指令和工程的 **uses** 指令的最后一行之间，插入关键字 **export**，后跟单元的访问过程或函数名称（在本例中，就是 **ListMoveF**），如果有多个窗体要重用，可列出多个函数或过程名；
 - ④ 删除 **begin** 和 **end** 之间的所有语句，让初始化代码块留空。
- EXAMDLL 工程的程序代码如下：

```
Library EXAMDLL;
```

```

uses
  SysUtils,
  Classes,
  Unit1 in 'Unit1.pas' {Form1};
exports
  ListMoveF;
{$R *.res}
begin
end.

```

(4) 生成 DLL 文件

执行【Project】→【Build EXAMDLL】命令，生成 EXAMDLL.DLL 文件。

(5) 编写一个应用程序调用该窗体

在 Delphi 集成环境中，新建一个应用程序，该应用程序的设计界面如图 10-7 所示。程序运行时单击【调用窗体】按钮，将出现如图 10-8 所示的可重用窗体，移动几个列表框到右边的列表框中，然后单击【确定】按钮，将会出现如图 10-9 所示的运行结果界面，界面上分别显示出两个列表框中的列表项数目。把该应用程序保存的项目文件名设置为“A_10_4”。



图 10-7 调用程序设计界面



图 10-8 弹出的可重用窗体

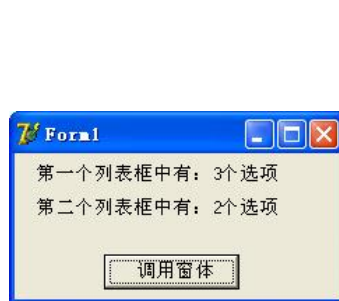


图 10-9 程序的运行结果界面

主要程序代码如下：

```

implementation
{$R *.dfm}
Function ListMoveF(var l1,l2 :Integer):WordBool;Far; External 'EXAMDLL.DLL'
procedure TForm1.Button1Click(Sender: TObject);
var
  l1,l2:Integer;
begin
  if ListMoveF(l1,l2) then
  begin
    Label1.Caption := '第一个列表框中有: '+inttostr(l1)+'个选项';
    Label2.Caption := '第二个列表框中有: '+inttostr(l2)+'个选项';
  end;
end;

```

10.2 典型实例

【实例题目】：显式调用 DLL 中的多种形式的函数

编写一个 DLL 文件, 取名为 FUNDLL4, 该库中有三个函数 AlphNum、DigNum 和 HuiWen, 它们的功能分别是求出字符中字母字符的个数、数字字符的个数和字符串是否为回文。然后编写一个项目文件, 在该项目文件中对这三个函数进行调用。要求: 在项目文件中采用显式调用的方法调用 DLL 中的函数。程序的设计界面与运行界面分别如图 10-10 和图 10-11 所示。程序执行时在 Edit1 编辑框中输入字符串, 接着在 ComboBox1 复选框中选择操作种类, 然后单击【执行操作】按钮将调用相应的函数, 得到执行结果。



图 10-10 程序设计界面



图 10-11 程序运行界面

【实现方法】

(1) 首先编写 DLL 文件, 包含三个函数: AlphNum、DigNum 和 HuiWen, 并把这三个函数在 exports 语句中进行声明。

(2) 再编写调用该 DLL 的应用程序。显式调用 DLL 中的函数的方法请参照前面的例子。

(3) 当调用的函数的参数类型与返回值类型都相同时, 为动态调用这些函数, 只需定义一种“函数类型”。但当需要动态调用的函数的返回值或参数类型不相同, 为每一类不相同的函数都应定义一种“函数类型”。

【界面设计】

调用程序的窗体组件属性设置及其作用如表 10-5 所示。

表 10-5 调用程序的组件属性设置及组件作用

组 件 名	属 性 名	属 性 值	作 用
Label1	Caption	'输入字符串'	
Edit1	Text	"	输入字符串
Label2	Caption	'请选择操作种类'	
ComboBox1	Items	字母字符个数 数字字符个数 判断是否回文	用来选择操作种类, 根据选择的操作种类调用相应的函数

续表

组 件 名	属 性 名	属 性 值	作 用
Label3	Caption	'操作结果'	
Edit2	Text	"	显示执行结果
Button1	Caption	'执行操作'	单击它将调用相应函数
Button2	Caption	'退出'	单击它将退出应用程序

【程序代码】

DLL（FUNDLL4）的程序代码如下：

```
library FUNDLL4;
uses
  SysUtils, Classes;
{$R *.res}
Function  AlphNum(Parastr:string):Integer;stdcall;      //统计字母字符的个数
var
  i,num:integer;
  t:string;
begin
  num:=0;
  For i:=1 to  length(Parastr) do
    begin
      t:=copy (Parastr,i,1);
      if (UpperCase(t)>='A')  and (UpperCase(t)<='Z')  then
        Num:=Num+1;
      end;
    Result:=num;
  end;
Function  DigNum(Parastr:string):Integer;stdcall;
var
  i,num:integer;
  t:string;
begin
  //统计数字字符个数，略
end;
Function  HuiWen(str:string):Boolean;stdcall;
var
  i,j:integer;
begin
  //判断 str 是否是回文，略
end;
exports
  DigNum,AlphNum,HuiWen;                                //声明可以被其他程序调用的函数
begin
end.
```

调用程序的主要程序代码如下：

```

implementation
{$R *.dfm}

type
  TIntFunction=Function(arr:String):Integer;stdcall;
  //该类型用来调用 AlphaNum 和 DigNum 函数
  TBoolFunction=Function(arr:String):Boolean;stdcall;
  //该类型用来调用 Huiwen 函数

procedure TForm1.Button2Click(Sender: TObject);
var
  Dllname:string;
  Hinst:THandle;
  ProcName:Pchar;
  Fpointer:TFatProc;
  MyFunctInt:TIntFunction;
  MyFunctBool:TBoolFunction ;
  k:integer;
begin
  getdir(0,Dllname);
  Dllname:=dllname+'FUNDLL4.dll';
  if ComboBox1.Text ='字母字符个数' then
  begin
    k:=1;  ProcName:='AlphaNum';
  end
  else
  if  ComboBox1.Text ='数字字符个数' then
  begin
    k:=2;  ProcName:='DigNum';
  end
  else
  begin
    k:=3;   ProcName:='HuiWen';
  end;
  Hinst:=SafeLoadLibrary(Dllname);
  if Hinst>0 then
  try
    Fpointer:=GetProcAddress(HInst,ProcName); //得到相应函数名的指针
    if Fpointer<>Nil then
    begin
      if(k=1) or (k=2)  then
      //如果 k=1 或 k=2, 是一种函数类型 (AlphaNum 和 DigNum)
      begin
        MyfunctInt:=TIntFunction(Fpointer);
        Edit2.text:=inttostr(MyfunctInt(edit1.text));
      end
      else
      //如果 k=3 是另一种函数类型 (huiwen)
      begin

```

```

MyfuncBool:=TBoolFunction(Fpointer);
if MyfuncBool(edit1.text) then
    Edit2.Text := '是回文'
else
    edit2.Text := '不是回文'
end;
end
else
    showmessage('DLL Function not found.');//显示函数没有找到信息
Finally
    FreeLibrary(Hinst);                //卸载 DLL
end
else
    ShowMessage(Dllname+' Library not Found');//显示 DLL 没有找到信息
end;

```

10.3 上机练习

【练习题目】：一维数组求值相关算法动态链接库的编制

编写一个 MyDLL.DLL 文件, 为该 DLL 创建四个函数 Max、Min、Sum 和 Aver, 分别用来求具有 N (参数) 个元素的一维数组的最大值、最小值、总和及平均值。然后再创建一个应用程序, 通过对这些函数进行显式调用 (又称动态调用) 来求产生的由 10 个两位随机数组组成的一维数组的最大值、最小值、总和及平均值。程序的设计界面如图 10-12 所示, 程序的运行界面如图 10-13 所示。程序运行时单击**【产生数组】**按钮将产生 10 个两位随机数并存放一个一维数组中, 同时把数组元素的值显示在第一个编辑框中。然后在组合框中选中相应的运算类型, 再单击**【计算】**按钮将根据运算类型得到相应的结果并显示在第二个编辑框中。



图 10-12 调用程序设计界面



图 10-13 调用程序运行界面

【要点提示】

首先编写 MyDLL 文件, 在该 DLL 中定义四个函数: Max、Min、Sum 和 Aver, 并在 exports 语句中加以说明, 然后执行**【Project】**→**【Build MyDLL】**命令以生成 DLL 文件。在调用程序中, 采用显式调用的方法来调用这些函数。

【参考代码】

DLL 程序代码如下:


```

library mydll;
uses
    SysUtils, Classes;
{$R *.res}
Function  Max(a:array of integer;n:integer):integer;stdcall;
var
    i,max_v:integer;
begin
    .....//此处求出参数数组 a 的最大值并作为函数值返回（略）
end;
Function  Min(a:array of integer;n :integer):integer;stdcall;
var
    i,min_v:integer;
begin
    .....//此处求出参数数组 a 的最小值并作为函数值返回（略）
end;
Function  Sum(a:array of integer;n:integer):integer;stdcall;
var
    i,Sum_v:integer;
begin
    .....//此处求出参数数组 a 的各元素之和并作为函数值返回（略）
end;
Function  Aver(a:array of integer;n:integer):integer;stdcall;
var
    i,Aver_v:integer;
begin
    .....//此处求出参数数组 a 的各元素平均值并作为函数值返回（略）
end;
exports
    Max,Min,Sum,Aver;
begin
end.

```

调用程序的主要程序代码：

```

implementation
type
    TintFunction=Function(arr:array of integer;n:integer):integer;stdcall;
var
    arr:array[1..10] of integer;
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
    i:integer;

```

```
begin
.....//产生十个两位随机数存放到一维数组 a 中，并显示在 Edit1 编辑框中（略）
end;
procedure TForm1.Button2Click(Sender: TObject);
var
    Dllname:string;
    Hinst:THandle;
    ProcName:Pchar;
    Fpointer:TFarProc;
    MyFunct:TIntFunction;
    k:integer;
begin
    getdir(0,dllname);
    dllname:=dllname+'\my.dll.dll';
    if ComboBox1.Text ='数组元素最大值' then
        ProcName:='Max'
    else
        if  ComboBox1.Text ='数组元素最小值' then
            ProcName:='Min'
        else
            if  ComboBox1.Text ='数组元素总和' then
                ProcName:='Sum'
            else
                if  ComboBox1.Text ='数组元素平均值' then
                    ProcName:='Aver';
                Hinst:=SafeLoadLibrary (Dllname);
                if Hinst>0 then
                    try
                        Fpointer:=GetProcAddress(Hinst,ProcName);
                        if Fpointer<>Nil then
                            begin
                                Myfunct:=TIntFunction(Fpointer);
                                k:=Myfunct(arr,10);
                                Edit2.Text :=inttostr(k);
                            end
                        else
                            showmessage('DLL Function not found.');
```

课后考场

一、选择题（20 分，每题 5 分）

1. _____不是使用动态链接库（DLL）的优点。
A. 便于开发大型软件 B. 代码重用
C. 独立于编程语言 D. 在内存中可能存放 DLL 的多个拷贝
2. Delphi 中，表示 DLL 项目的关键字是_____。
A. Program B. Project C. Library D. DLL
3. 在显式调用 DLL 中的函数时，_____Windows API 函数不会被用到。
A. LoadLibrary B. FreeLibrary
C. LibraryExec D. GetProcAddress
4. 关于 DLL 的说法，不正确的是_____。
A. DLL 是实现 Windows 应用程序代码重用和共享的重要手段
B. 一个 DLL 文件中可以包含多个过程或函数
C. 利用 DLL 可以实现窗体的重用
D. DLL 就是 C 等高级语言的函数库在 Windows 编程环境下的另一个说法

二、填空题（40 分，每空 5 分）

1. DLL 是在应用程序_____期间被链接进来的，而一般函数库是在应用程序使用链接器（Linker）_____文件时被链接进来的。
2. 传统的库函数是通过_____链接链接到应用程序中的，而 DLL 的链接方式是_____链接，即在程序运行时动态地加载所需的函数。
3. 在 DLL 项目中，编写若干个函数，若希望这些函数能被其他应用程序调用，则应在_____语句中进行声明。
4. 要生成 DLL，应执行【_____】→【Compile】/【Build】命令。
5. 动态载入方式是指在编译之前并不知道将会调用哪些 DLL 函数，完全是在运行过程中根据需要决定应调用哪些函数，并用 LoadLibrary 加载 DLL 到内存，用_____动态获得 DLL 函数的入口地址。当一个 DLL 文件用 LoadLibrary 显式加载后，在任何时刻均可以通过调用_____函数显式地从内存中把 DLL 给卸载。

三、程序设计题（40 分，每题 20 分）

1. 编写一个 DLL，该 DLL 中包含有一维数组的选择法排序、冒泡法排序、一维数组的反序存放几个过程，这些过程均能被其他应用程序调用。
2. 编写一个应用程序，用来调用题 1 编写的 DLL 中的几个过程，要求使用显式调用的方法。（先把题 1 生成的 DLL 文件复制到本应用程序所在目录）

第 11 章 组件开发技术

本章要点

- ▮ 组件的概念
 - ▮ 开发组件的一般步骤
 - ▮ Delphi 中的相关类
 - ▮ 创建组件单元的方法
 - ▮ 为组件添加属性、事件和方法
 - ▮ 组件的调试与安装
-

11.1 理论知识

11.1.1 组件与组件技术概述

1. 组件概念

使用 Delphi 7 进行程序设计时,时时刻刻都要和组件打交道。组件是 Delphi 应用程序的程序构件, Delphi 在发布时就提供了许多优秀的组件。另外,用户还可以购买或下载第三方开发的组件并应用在自己的应用程序中,同时还可以根据组件的标准自己开发组件。

组件是模块化程序设计方法发展到一定阶段的产物,在软件工程的角度来考虑,开发者总是希望把一个庞大的应用程序划分成多个模块。其中,每个模块都保持一定的功能独立性,在协同工作时,通过相互之间的接口来完成实际的任务。每一个这样的模块称为组件,一个设计良好的应用系统往往被切分成多个组件,这些组件可以单独开发、单独编译,甚至单独调试和测试。当所有的组件开发完成后,把它们组合在一起就得到了完整的应用系统。当系统的外部软硬件环境发生变化或者用户的需求有所变更的时候,并不需要对所有的组件进行修改,而只需要对受影响的组件进行修改,然后重新组合就可得到新的升级软件。

从用户的角度来看,组件是一个“黑匣子”,用户不必关心组件是如何完成任务的,它只要能完成用户要求的功能即可。从组件编写者的角度看,组件就是 Object Pascal 中类的实例,但类与组件并不是等同的,所有的组件都是类,但并不是所有的类都是组件。

2. 从一个实际的组件看组件的组成

下面通过一个 Delphi 提供的 TButton 组件的声明来分析组件的结构。TButton 组件的声明如下(中间省略了一些代码,已用“……”表示):

```
TButton = class(TButtonControl)
```

```
private
    FDefault: Boolean;
    FCancel: Boolean;
    FActive: Boolean;
    FModalResult: TModalResult;
    procedure SetDefault(Value: Boolean);
    .....
protected
    procedure CreateParams(var Params: TCreateParams); override;
    procedure CreateWnd; override;
    procedure SetButtonStyle(ADefault: Boolean); virtual;
public
    constructor Create(AOwner: TComponent); override;
    procedure Click; override;
    function UseRightToLeftAlignment: Boolean; override;
published
    .....
    property Cancel: Boolean read FCancel write FCancel default False;
    property Caption;
    property Constraints;
    property Default: Boolean read FDefault write SetDefault default False;
    .....
    property ModalResult: TModalResult read FModalResult write FModalResult default 0;
    .....
    property TabStop default True;
    .....
    property OnClick;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    .....
end;
```

分析该组件声明，可以得到以下结论。

(1) 组件的声明

组件的声明其实就是类的声明。代码的一开始用 **class** 声明了一个名为 **TButton** 的类，该类的基类是 **TButtonControl**。类的成分有四种，分别是 **private**、**protected**、**public** 和 **published**。其中 **private** 中的成员只能被本类中的成员函数或过程使用；**protected** 成员不但可以被该类的成员函数或过程使用，而且可以被该类的子类使用；**public** 成员可以不受限制地被外界访问；**published** 成员则显示在对象观察器中，用户可以在设计时进行修改。

(2) 方法

在组件中用 **Procedure** 或 **Function** 定义的过程或函数均是方法，方法可以被外界调用。

(3) 属性

在组件的定义中需要特别注意的是 **property** 关键字，该关键字用于组件属性的声明，声

明属性时可用 `read` 关键字标识了属性的读取方法，用 `write` 关键字标识属性的赋值方法。如语句：

```
property Default:Boolean read FDefault write SetDefault default False;
```

的含义为：声明一个名为 `Default` 的属性，该属性从私有成员变量 `FDefault` 读取属性值，使用 `SetDefault` 函数给属性赋值，且属性默认值为 `False`。

（4）事件

事件是一类特殊的属性，实际上它是一个函数或过程的指针，也可以是系统指供的特定的名称。如语句：

```
property OnMouseDown;  
property OnMouseMove;
```

声明的就是事件，事件 `OnMouseDown` 在鼠标按下时发生，事件名 `OnMouseMove` 在鼠标移动时发生。

3. 开发组件的一般步骤

创建一个组件，大致可以分为以下 6 个步骤。

- （1）确定一个基类。
- （2）创建一个组件单元。
- （3）在新组件中添加属性、方法和事件。
- （4）测试该组件。
- （5）在 `Delphi` 中注册该组件。
- （6）为该组件建立帮助文件。

本章只讨论前 5 个步骤，帮助文件的制作对商业化的组件来说非常重要，但不是本章讨论的内容，有兴趣可参阅相关的 `Delphi` 资料。

11.1.2 确定组件基类

1. Delphi 组件结构

创建组件的第一步就是确定组件的基类，要确定组件的基类，就必须首先熟悉 `VCL` 的继承关系，以及在不同层次的继承关系上类的不同用途。`Delphi` 中组件的创建是与 `VCL` 类库紧密相关的，只有对 `VCL` 类层次结构有了深刻的理解后，才能编写出良好的自定义组件。

图 11-1 是 `Delphi` 帮助文件中提供的组件结构图。

可见，`Delphi` 的类继承结构是典型的单根结构，`TObject` 抽象类是所有类的祖先，程序员无法创建不继承于任何类的类。作为组件开发人员不要从 `TObject` 直接派生自定义组件，选择一个好的基类将会使组件开发事半功倍。

一般来说，可视组件和非可视组件都是从 `TComponent` 类派生的，开发组件应该选择 `TComponent` 及其子类为组件基类，可视组件的基类 `TControl` 从 `TComponent` 派生，它增加了对 `Windows` 的绘制能力的支持，可以在屏幕上显示图形和文字，但开发可视组件时一般不选择 `TControl` 为组件基类，而是选择 `TControl` 的子类为组件基类，因为 `TWinControl` 和 `TGraphicControl` 能够更好地支持可视组件的实现。

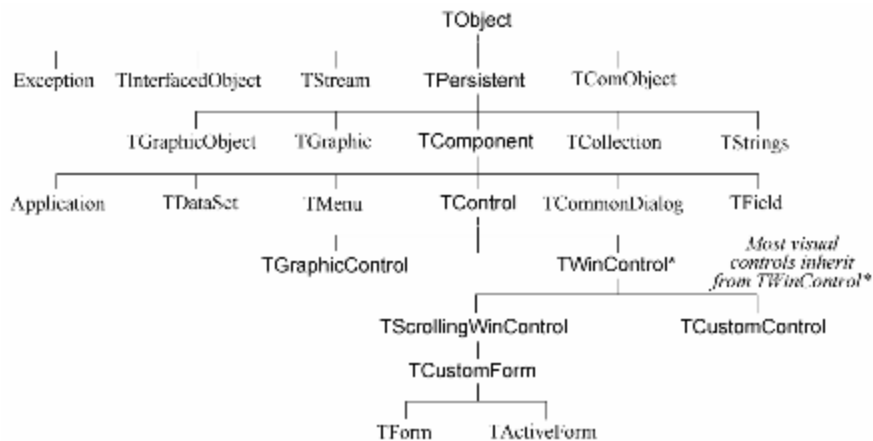


图 11-1 Delphi 的组件结构图

2. 几种常用的基类

(1) TWinControl 类

程序中经常使用的编辑框、选择框和列表框等 Windows 组件都是从 TWinControl 类派生的，这类组件常用于与用户交互，不但可以显示和处理数据，而且可以接收用户的输入。

TWinControl 类的特征如下。

- 有窗口句柄。句柄是 Windows 中对象的一个 32 位数字标识，每个 Windows 窗口都有惟一的句柄，操作窗口的方法都需要传递这个句柄来指明操作对象，Delphi 封装了 Windows 的窗口，隐藏了句柄的处理细节，使开发者不需再关心句柄的处理，TWinControl.Handle 属性记录了组件的句柄，在使用某些 Windows API 函数时需要使用它。
- 能够接受输入焦点。获得焦点的组件可以感知和处理用户的键盘和鼠标的操作，并对这些操作进行处理，实现程序与用户的交互。
- 能够作为其他组件的父组件。组件的父组件是组件在窗体上显示和进行其他相关处理的基础，在显示的时候总是根据它的父组件来确定显示位置。一个父组件可以看做一个容器，在它被销毁时会自动销毁它的所有子组件，可以使用 TWinControl.ControlCount 获得子组件的个数和 TWinControl.Controls 遍历全部子组件。

(2) TGraphicControl 类

TGraphicControl 是图形组件类，也属于可视组件，但它与 TWinControl 有很大不同，它没有句柄，不能获得焦点，也不能作为其他组件的父组件。这类组件的缺点是难以和用户交互，其优点是节省系统资源，而且有较快的图形绘制速度。一般在希望开发一个仅用于显示的组件时，可选择 TGraphicControl 或者它的子类为组件基类。

TGraphicControl 的 Canvas 属性为组件的绘制提供了方便，可以直接使用画布绘制组件的外观，如果要从 TGraphicControl 派生新组件，它的 Paint 方法必须被重载，以绘制组件外观。

(3) TCustomControl 类

常用的可视组件一般都从以 TCustom 开头的类派生，如 TEdit 组件是从 TCustomEdit 类派生的、TListView 是从 TCustomListView 派生的，等等。实际上在 TCustom 类中已经实现了

组件的属性、方法和事件，只不过它们的成员都是 `protected` 时，即对于外部是不可见的，无法被使用，而在它的派生类里才将这些属性和方法公开。

这种组件实现的模式对组件的开发比较有利，一般在选择基类时，更多的是选择带有 `TCustom` 开头的类，这样便于控制属性的可见性、修改或限制原有属性的实现，以及避免新属性与已有属性冲突。当然，有时只想添加一些功能而不想限制或者修改组件时，也可以从不带 `TCustom` 的类开发组件，这要根据具体情况来定。

下面通过开发一个实际的剪贴板文本查看组件来说明组件的开发过程。剪贴板查看组件具有以下功能。

(1) 该组件的基类是 `TPanel` 组件，能够通过属性设置它的标题和是否可见，组件名为“`ClipBoardViewer`”。

(2) 当剪贴板中存放的是文本信息且当文本发生变化时，将发生一个名为 `ClipBoardChangeText` 的事件，该事件有一个参数 `Text` 用来传递剪贴板上的数据。

(3) 为该组件添加一些常用类型的属性，包括简单类型、枚举类型、集合类型、对象类型、数组类型等。

11.1.3 创建组件单元

用户可以自己编写一个实现组件类功能的组件单元，但这比较麻烦。因此也可以通过菜单来创建新组件单元，创建的步骤如下。

(1) 执行【**Component**】→【**New Component**】菜单命令。将会出现如图 11-2 所示的【**New Component**】对话框。

(2) 指定组件的基类。指定方法是在【**Ancestor type**】列表框中选择相应的基类名，本例选择 `TCustomPanel`。

(3) 设定组件名。在【**Class Name**】文本框中输入类名，本例输入的类名为“`TclipBoardViewer`”。

(4) 设定组件出现在哪一个选项卡中。在【**Palette Page**】列表框中可以选择一个选项卡，本例选择【**Samples**】选项卡，该组件注册过后将会出现【**Samples**】选项卡中。

(5) 指定组件单元的保存位置和名称。在【**Unit file name:**】文本框中输入组件单元存放的路径和单元文件名，单元文件名和类名相同（不包含开头的字母 `T`）。

(6) 自动创建组件单元。设置完成后如图 11-3 所示，此时单击【**OK**】按钮系统将自动创建组件单元。创建的组件单元的代码如下：

```
unit ClipBoardViewer;
interface
uses
  SysUtils, Classes, Controls, ExtCtrls;
type
  TClipBoardViewer = class(TCustomPanel)
  private
    { Private declarations }
  protected
```



```

    { Protected declarations }
public
    { Public declarations }
published
    { Published declarations }
end;
procedure Register;
implementation
procedure Register;
begin
    RegisterComponents('Samples', [TClipBoardViewer]);
end;
end.

```

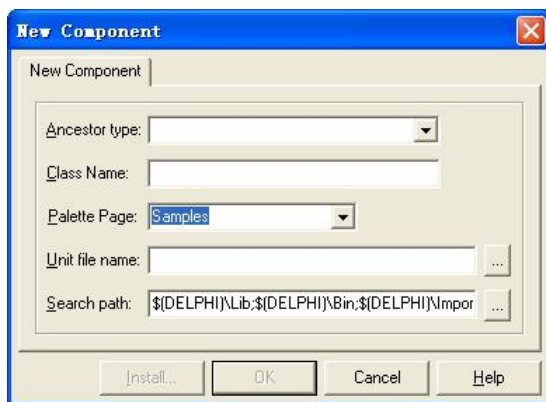


图 11-2 【New Component】对话框

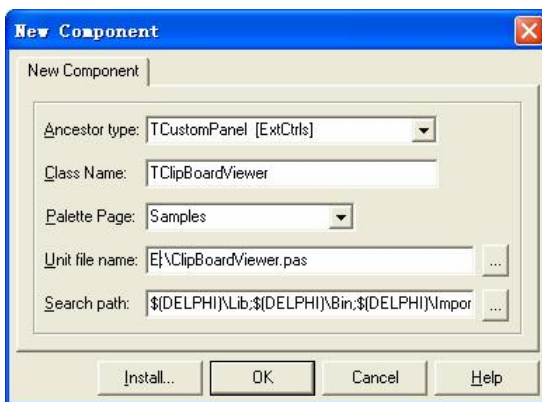


图 11-3 剪贴板文字查看组件的设置

代码中的 RegisterComponents 过程的作用是注册组件。

11.1.4 创建包工程

包工程是组件的容器，组件一般由包工程来安装和发布。包工程的主要用途就是封装 Delphi 的组件，组件包工程的扩展名为.dpk，Delphi 自带的组件都打包在 Delphi 的 Lib 目录的 Delusr.dpk 文件中。

下面创建一个包工程用来容纳并测试创建的 TClipBoardViewer 组件，打包的步骤如下。

(1) 执行【File】→【New】→【Other】命令，在出现的【New Items】对话框中选中【Package】图标，然后单击【OK】按钮，此时将会出现如图 11-4 所示的【Package】包工程管理器窗口。

(2) 单击【Add】按钮，将会出现如图 11-5 所示的【Add】对话框，在该对话框的【Unit file name】文本框中输入组件单元的文件名或通过单击【Browse】按钮来选择一个组件单元文件名。输入或选择“E:\ClipBoard Viewer.pas”。

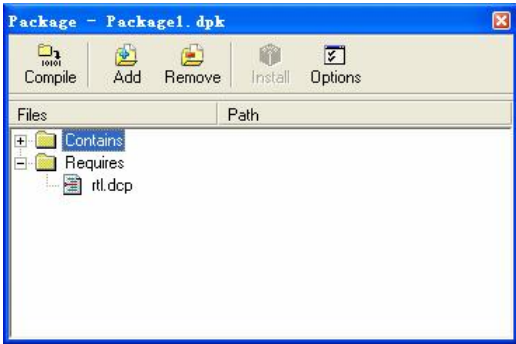


图 11-4 包工程管理器

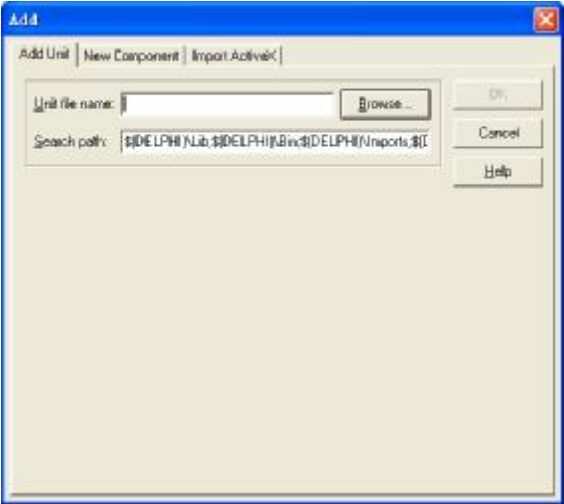


图 11-5 【Add】对话框

(3) 保存包工程，工程名文件为“Package1.dpk”。

(4) 单击【Compile】按钮对组件单元进行编译，若无错误则编译并注册组件。也可以直接单击【Install】按钮来注册组件。

组件注册后，就可以使用它了。使用方法是再新建一个窗体，然后在【Sample】选项卡中找到 ClipBoardViewer 组件，如图 11-6 所示。在窗体上绘制一个该组件，如图 11-7 所示。此时就可以像使用 Delphi 的其他组件一样来使用该组件了。



图 11-6 ClipBoardViewer 组件



图 11-7 添加到窗体上的 ClipBoardViewer 组件

11.1.5 在组件中添加属性

1. 属性的类型

属性是访问组件内部存储域的接口。通过属性，组件使用者能够修改或读取存储域的值。

组件的属性一定具有某种类型，Object Pascal 语言中的数据类型规则同样适用于属性，属性的类型决定了它的值在对象观察器中如何设置。属性的类型如表 11-1 所示。

表 11-1 属性类型及其说明

属 性 类 型	类 型 说 明
简单	包括整型、浮点型、实型、字符型和字符串型等基本类型，在对象观察器中可以直接输入属性的值
枚举	枚举类型（包括 Boolean）的属性在源代码中定义它的值。在对象观察器中，可以通过双击属性值栏来查看所有可能的值并进行设置，也可以从其下拉列表中浏览全部可能的值并设置

续表

属 性 类 型	类 型 说 明
集合	可以展开集合类型的属性，并把每一个集合元素看成一个 Boolean 型：值为 True 时，表示集合包含这个元素，值为 False 时，表示集合不包含该元素
对象	对象属性必须由 TPersistent 类派生。编辑对象类型的属性常常需要有它们自己的属性编辑器。然而，如果对象本身的属性也是公开的，则在对象观察器中允许展开这个对象类型的属性并分别编辑它们
数组	数组属性必须有它们自己的属性编辑器，在对象观察器中无法编辑这种类型的属性，数组类型的属性是在类的 Public 部分声明的

一般来说，声明一个属性需要描述三个对象：属性名、属性类型、读或设置属性值的方法，有时还可以设置属性的默认值。

属件可以定义在类的 **public** 部分和 **published** 部分，在 **public** 部分定义的属性只能在程序中访问，在 **published** 部分定义的属性将显示在对象观察器中，可以在设计时对其属性值进行设置。

2. 简单属性的声明

下面为组件创建三个属性 **BorderWidth**、**Caption** 和 **Visible**，其中 **BorderWidth** 的类型为整型，其他两个属性是系统属性，用来设置标题和可见性。添加了这三个属性后，组件的主要代码如下：

```

type
  TClipBoardViewer = class(TCustomPanel)
  private
    { private declarations }
    FBorderWidth:Integer;
    procedure SetBorderWidth(Value:Integer);
  protected
    { protected declarations }
  public
    { public declarations }
  published
    { published declarations }
    property BorderWidth:Integer
      read FBorderWidth write SetBorderWidth Default 0;
    property Caption;
    property Visible;
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('Samples', [TClip BoardViewer]);
end;
procedure TClip BoardViewer.SetBorderWidth(Value:Integer);
begin
  FBorderWidth:=Value;

```

```

Realign;
Invalidate;
end;

```

代码编写好后，在包工程中编译该组件，然后新建一个工程，在窗体上添加一个 TClipBoardViewer 组件，选中该组件，对象观察器窗口如图 11-8 所示。

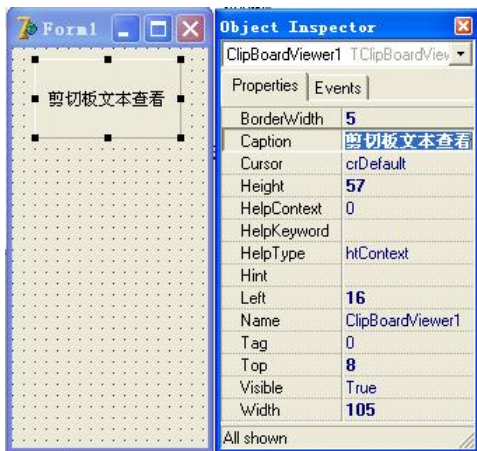


图 11-8 添加若干个属性后的对象及对象观察器

注意：对于用户定义的属性来说，一般要在类的 private 部分定义一个存储该属性值的变量，变量名一般以 F 开头，如本例的“FBorderWidth”。

3. 枚举型和集合型属性的声明

要为组件增加枚举型或集合型属性，首先应定义相应的枚举类型或集合类型，然后用该类型来定义属性，定义方法与简单类型基本相同。下面为 ClipBoardViewer 添加两个属性 WeekDay 和 WorkDay，WeekDay 是一个枚举类型属性，表示一个星期中的某一天，WorkDay 表示工作日，是 WeekDay 类型的集合类型属性。添加了这两个属性后，组件的主要代码如下。

```

type
  TWeekDayType=(Sunday,Monday,Tuesday,Wednesday,Tuesday,Friday,Saturday);
  TworkdayType=Set of TweekDaytype;
  TClipBoardViewer = class(TCustomPanel)
  private
    FBorderWidth:Integer;
    FWeekDay :TWeekDay Type;
    FWorkDay : TWorkDayType;
    procedure SetBorderWidth(Value:Integer);
  protected
    { protected declarations }
  public
    { public declarations }
  published
    { published declarations }

```

```

property BorderWidth:Integer
    read FBorderWidth write SetBorderWidth Default 0;
property Caption://:String Read Fcaption Write Fcaption;
property Visible;
property Weekday:TweekDayType Read  FWeekDay write Fweekday;
property Workday:TworkDayType Read  FWorkDay write Fworkday;
end;

```

本例在组件类声明的前面定义了枚举类型 `TWeekDayType` 和集合类型 `TWorkDayType`，它们分别是枚举属性 `WeekDay` 和集合属性 `WorkDay` 的属性类型。在包工程中重新编译组件后，在对象监视器里可以看到这两个属性。枚举类型属性会自动维护一个列表，可以在设计时选择枚举项目，如图 11-9 所示。集合类型属性会在左边出现一个“+”，单击它可以打开集合的元素列表，每个元素都可以选择 `True` 或 `False` 来设置集合属性是否包含该元素，如图 11-10 所示。

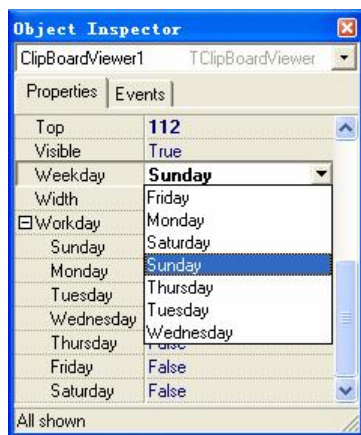


图 11-9 枚举型属性的设置

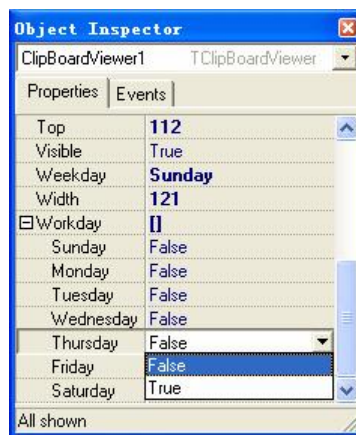


图 11-10 集合型属性的设置

4. 数组属性的声明

可以为组件添加数组属性，数组属性的实现与简单类型属性的实现方法有一些不同，给属性设置值其实是设置数组元素的值，读取属性的值其实是读取数组元素的值，为标识读写的是数组属性的哪一个元素，读写方法中应增加一个表示数组元素下标的参数。

下面为 `TClipboardViewer` 组件添加一个整型数组属性 `IntArr`，该属性具有 10 个元素。添加该属性后，组件的代码如下：

```

type
    //.....省略的部分代码
    TClipboardViewer = class(TCustomPanel)
    private
        //.....省略的部分代码
        FIntarr :Array[0..10] Of integer;           //存储数组属性的值域
        Function GetIntArr(i:integer):Integer;
        procedure SetIntArr(i,value:integer);

```

```

protected
public
    property IntArr[i:Integer]:Integer read GetIntArr Write SetIntArr;    //属性声明
published
    //.....省略的部分代码
end;
procedure Register;
implementation
    //.....省略的一些过程代码
Function TClipBoardViewer.GetIntArr(i:integer):integer;
begin
    result:=FIntArr[i];
end;
procedure TClipBoardViewer.SetIntArr(i, Value:Integer);
begin
    FIntArr[i]:=Value;
end;

```

用以上代码可见，数组属性的读写方法均增加了一个参数 *i*，用来表示要使用的数组元素的下标；数组属性定义在类声明的 **public** 部分，因此数组属性无法在对象观察器中编辑，只能在程序运行中通过程序代码来访问。

下面使用该组件给数组属性的每一个元素赋一个随机的两位数，然后求数组属性各元素值的平均值，步骤如下。

- (1) 重新在包工程中编译组件单元。
- (2) 新建一个工程，在窗体 **Form1** 上添加该组件，然后再添加一个各为 **Button1** 的按钮组件。设置 **Button1** 的标题为“求平均”。
- (3) 在窗体上添加一个名为 **Label1** 的标签组件，用来显示求得的平均值。
- (4) 编写按钮组件的 **OnClick** 事件代码，如下所示：

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i:integer;
    Aver:Real;
begin
    Randomize;
    Aver:=0;
    For i:=0 to 9 Do
        begin
            ClipBoardViewer1.IntArr[i]:=10+Random(90);
            Aver:=Aver+ ClipBoardViewer1.IntArr[i] ;
        end;
    Aver:=Aver/10;
    Label1.Caption:='十个随机数的平均值: '+FloatToStr(Aver);
end;

```

(5) 运行程序，单击 Button1 按钮，程序的执行结果如图 11-11 所示。



图 11-11 数组属性演示程序执行结果

5. 对象属性的声明

组件的属性还可以是对象。对象的使用比较复杂，在使用之前要创建对象，在使用过后要释放对象，这都要在组件中自动实现。一般是在类的构造器中创建对象，在析构器中释放对象。

下面为 TClipBoard Viewer 组件添加一个名为 Imagepic 的属性，该属性的类型是 TBitmap 型。添加该属性后，组件的代码如下：

```
type
//.....省略了类型定义代码
  TClipBoardViewer = class(TCustomPanel)
  private
//.....省略了私有变量定义
    FImagePic:TBitmap;
//.....省略了一些过程说明
    procedure SetImagePic(Pic:TBitmap);
  protected
  public
    Constructor   Create(Comob:TComponent);override;
    Destructor    Destroy;Override;
    property IntArr[i:Integer]:Integer read GetIntArr Write SetIntArr;
  published
//.....省略了一些属性说明
    property Imagepic:TBitmap Read FImagePic Write SetImagePic;
  end;
  procedure Register;
  implementation
//.....省略的过程代码见前文
  Constructor   TClipBoardViewer.Create(comob:TComponent);
  begin
    inherited;
    FImagepic:=TBitmap.create;
  end;
  Destructor   TClipBoardViewer.Destroy ;
  begin
    if assigned(FImagePic) then
```

```

FImagePic.Free;
    inherited;
end;
procedure TClipBoardViewer.SetImagepic(Pic:TBitmap);
begin
    if assigned(FImagePic) then
        FImagePic.Free;
        FImagePic:=Pic;
end;

```

11.1.6 在组件中添加事件

1. 事件的来源

事件是一种特殊的属性，当某个动作发生时，就会执行预先指定的代码。事件可能是由用户交互操作、操作系统、程序代码等产生。事件的机制就是事件与相应的代码相联系，当事件发生时，就会调用相应的代码。事件与代码的链接称为事件属性，以方法指针的形式提供。而这样的代码就是处理事件的方法，称之为事件处理过程。

例如，当用户单击鼠标，一个 WM_MOUSEBUTTONDOWN 消息就会被发送到 Win32 系统。Win32 系统就会把这个消息传递给预定义的组件，该组件就会响应该消息。该组件首先检查是否有相应的执行代码。如果有的话，就执行这段代码，即执行事件处理过程。

OnClick 事件是 Delphi 的一种标准事件属性。OnClick 事件或者其他事件都有一个相应的事件调度方法。事件调度方法通常是在组件的 `protected` 部分声明，由它来检查事件属性是否指向了用户自定义组件提供的代码。

作为组件编写者，需要编写事件声明、事件属性及事件调度的所有代码。组件使用者只需建立事件处理过程，然后调用事件调度方法检查处理事件的代码是否存在。若存在，就执行它。

2. 确定事件类型

在添加事件之前，应当确定事件的类型，也就是确定事件处理函数的参数表，合理的参数可以使事件处理函数的实现大大简化。Delphi 为 VCL（Visual Class Library，可视类库）组件预先定义了许多事件类型，表 11-2 列出了一些常用的事件类型，这些事件类型的第一个参数都是 `Sender`。

表 11-2 常用的 Delphi 预定义事件类型

事 件 类 型	说 明
TNotifyEvent	通知事件类型，是大多数事件的类型，只有一个 <code>Sender</code> 参数
TMouseEvent	鼠标事件类型，会传递鼠标位置、按键等信息
TKeyEvent	键盘事件类型，传入按下键的代码
TCloseQueryEvent	关闭事件类型，允许给 <code>CanClose</code> 参数赋值以控制窗体是否关闭
THelpEvent	帮助事件类型
TdragDropEvent	拖动事件类型，会给出拖动源和拖动目标等信息
TcanResizeEvent	组件大小变化事件，允许给 <code>Resize</code> 参数赋值控制是否改变组件大小

这些事件类型中用得比较多的是 `TNotifyEvent`。该事件只通知组件发生了某个事件，且没有参数传递。

用户也可以自定义事件类型，比如，要实现在系统剪贴板文本变化时触发的事件，应该考虑程序在事件发生时可能需要使用的参数来确定事件类型，可定义事件类型为 `TClipboardTextChangeEvent`，声明如下：

```
TClipboardTextChangeEvent = Procedure(Sender:TObject;Text:String) of Object;
```

其中参数 `Text` 用来存放剪贴板中传来的内容。

3. 加入事件

加入事件的方法与属性的声明基本一致。例如，给要加入 `TClipboardViewer` 组件添加 `TClipboardTextChangeEvent` 事件，代码如下：

```
TClipboardTextChangeEvent=Procedure(Sender:TObject;Text:String) of Object;
TClipboardViewer = class(TCustomPanel)
private
    FOnClipboardText:TClipboardTextChangeEvent;
//.....
published
    property OnClipboardText:TClipboardTextChangeEvent read FOnClipboardText
        Write FOnClipboardText;
end;
```

4. 触发事件

在组件中使用标准事件时，不需要考虑触发事件，而由系统自动触发。对某些事件，触发条件是必须的。例如，一个 `MouseDown` 事件是在用户按下鼠标的左键时发生，`Windows` 给应用程序发送 `WM_LBUTTONDOWN` 消息。接到消息后，一个组件调用它的 `MouseDown` 方法，再调用用户的 `OnMouseDown` 事件处理过程代码。但是有些事件却不是那么容易描述清楚的。例如，滚动条有一个 `OnChange` 事件，可被各种情况触发，包括按键、鼠标单击等。当定义事件时，开发者必须保证定义的事件能被正确地调用。

下面为 `TClipboardViewer` 组件定义一个事件 `ClipboardTextChangeEvent`，该事件的触发消息是 `WM_DRAWCLIPBOARD`。`TClipboardViewer` 组件的完整代码如下：

```
unit ClipboardViewer;
interface
uses
    Windows,Messages,SysUtils, Graphics,Classes, Controls, ExtCtrls,Clipbrd;
type
    TClipboardTextChangeEvent=Procedure(Sender:TObject;Text:String) of Object;    //事件类型
    TWeekDayType=(Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday);
    TworkdayType=Set of TweekDaytype;
    TClipboardViewer = class(TCustomPanel)
private
    { private declarations }
//.....省略的代码见前文
```

```

FNextClipBoardViewer:HWND;           //用来记录监视剪贴板的下一个程序
FHandle:HWND;                         //用来记录组件句柄
FOnClipBoardText:TClipBoardTextChangeEvent; //事件的私有变量
//.....省略的代码见前文
procedure WMDrawClipBoard(var Msg:TMessage);message WM_DRAWCLIPBOARD;
procedure WMChangeCBChain(Var Msg:TMessage);
    Message WM_CHANGECHAIN;           //当发出相应的消息时，激发的过程
protected
public
    Constructor Create(Comob:TComponent);override;
    Destructor Destroy;Override;
    property IntArr[i:Integer]:Integer read GetIntArr Write SetIntArr;
published
    property BorderWidth:Integer
        read FBorderWidth write SetBorderWidth Default 0;
    property Caption;
    property Visible;
    property Weekday:TweekDayType Read FWeekDay write Fweekday;
    property Workday:TworkDayType Read FWorkDay write Fworkday;
    property Imagepic:TBitmap Read FImagePic Write SetImagePic;
    property OnClipBoardText:TClipBoardTextChangeEvent read FonClipBoardText
        Write FonClipBoardText;
end;
procedure Register;
implementation
procedure Register;
begin
    RegisterComponents('Samples',[TClipBoardViewer]);
end;
//.....相关方法代码已经省略
Constructor TClipBoardViewer.Create(comob:TComponent);
begin
    inherited;
    FImagepic:=TBitmap.create;
    if not(csDesigning in ComponentState) then
        begin
            FNextClipBoardViewer:=SetClipBoardViewer(Handle);
            //注册本组件为剪贴板剪切程序
            FHandle:=Handle;
        end;
    end;
end;
Destructor TClipBoardViewer.Destroy ;
begin
    if assigned(FImagePic) then
        FImagePic.Free;
    if not(csDesigning in ComponentState) then

```

```

        ChangeClipBoardChain(FHandle,FNextClipBoardViewer);//取消剪贴板监视
    inherited;
end;
procedure TClipBoardViewer.WMChangeCBChain(var Msg:TMessage);
begin
    if FNextClipBoardViewer<>0 then
        SendMessage(FNextClipBoardViewer,Msg.Msg,Msg.Wparam,msg.Lparam);
end;
procedure TClipBoardViewer.WMDrawClipBoard(var Msg:TMessage);
begin
    if assigned(FOnClipBoardText) AND ClipBoard.hasFormat(CF_TEXT) then
        FOnClipBoardText(self,ClipBoard.asText);
    if FnextClipBoardViewer<>0 then
        SendMessage(FNextClipBoardViewer,Msg.msg,Msg.Wparam,msg.Lparam);
end;

```

注意：在上面的代码中，组件在创建后用 `SetClipboardViewer` 函数将注册自己为剪贴板监视程序，这样在系统剪贴板发生变化时会发送 `WM_DRAWCLIPBOARD` 消息给组件，在这个消息的处理函数 `WMDrawClipBoard` 中，调用了 `FOnClipBoardText` 函数，也就是触发了事件。在触发事件后把系统剪贴板变化消息向下发送。

11.1.7 调试组件

下面编写一个项目来测试编写的 `TClipBoardViewer` 组件。新建一个项目，在该项目中添加一个 `Memo` 组件和一个 `TClipBoardViewer` 组件，程序的设计界面如图 11-12 所示。然后编写 `TClipBoardViewer` 组件的 `ClipBoardText` 事件代码，如下所示：

```

procedure TForm1.ClipBoardViewer1ClipBoardText(Sender: TObject;
    Text: String);
begin
    Memo1.Text :=Text;
end;

```

程序执行时，任意选中一些文本，然后按 `Ctrl+C` 键，程序的运行界面如图 11-13 所示。



图 11-12 程序设计界面



图 11-13 程序运行界面

11.1.8 制作组件图标和发布组件

1. 制作组件图标

组件创建好后, 应为组件提供图标。可以通过 Delphi 工具 Image Editor 为组件创建图标, 创建的图标是一个 24×24 的资源文件, 其后缀名为 RES, 应改为 DCR, 并且组件图标的颜色也最好不要超过 16 色。

创建了一个位图后, 必须给这个位图命名。位图的名称要跟组件的类名相同, 而且要大写。位图文件必须与组件的单元文件位于同一个目录下, 编译这个单元时, 位图资源会被自动加到组件库中。

例如, 为 TClipboardViewer 组件创建的 24×24 像素的位图, 名称应为 CLIPBOARDVIEWER.DCR, 并且要与 TClipboardViewer 组件放在同一个目录下。

2. 发布组件

组件的最大优点就是有良好的共享性, 做好的组件可以由发布到网上, 供大家使用。发布组件有以下两种方法。

(1) 发布组件及其源码

将组件的包工程及组件单元文件、资源文件等全部发布, 用户可以用 Delphi 打开扩展名为 *.dpk 包工程, 然后编译安装组件即可。

(2) 只发布组件

如果不想发布组件源码, 则需要发布以下文件: 组件实现单元的编译结果文件 (*.dcu), 资源文件 (*.DCR 和 *.dfm), 以及包的库文件 (*.bpl) 和包的符号文件 (*.dcp), 这两个文件一般在 Delphi 目录下的 Projects\bpl 目录里。

发布组件时还应当附加一个组件的帮助文件和例程, 以便于用户理解和使用。

11.2 典型实例

【实例题目】: 支持文件拖放操作的列表框组件

编写一个名为 TDropFileListBox 的列表框组件, 该组件能够支持文件的拖放操作, 即当把磁盘上的磁盘文件名拖动到列表框中时, 磁盘文件名将作为列表项自动添加到列表框中。

【实现方法】

(1) 为实现文件拖放, 涉及如下三个 API 函数。

! DragAcceptFiles(): 初始化某窗口使其允许或禁止接受文件拖放。

! DragQueryFile(): 查询拖放的文件名。

! DragFinish(): 释放拖放文件时使用的资源。

(2) 文件拖放操作实现的基本思路如下: 首先调用 DragAcceptFiles() 函数初始化组件窗口, 使其允许接受文件拖放, 然后等待 WM_DropFiles 消息 (一旦用户进行了拖放文件操作, 组件窗口即可获得此消息), 获得消息后即可使用 DragQueryFile() 函数查询被拖放的文件名, 最后调用 DragFinish() 释放资源。

(3) 新组件的功能。从 TListBox 为组件的基类建立组件, 新组件名为: TDropFileListBox,

它比标准 TListBox 增加了一个 OnDropFiles 事件和一个 DropEnabled 属性。当 DropEnabled 为 True 时即可接受文件拖放，文件拖放完成后激发 OnDropFiles 事件，该事件提供一个 FileNames 参数让用户获得文件名。

【程序代码】

```
unit DropFileListBox;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ShellApi;
  //增加 ShellApi 单元，因为用到的三个 API 函数声明在该单元文件中
type
  TMyNotifyEvent = procedure (Sender: TObject; FileNames: TStringList) of object;
  //自定义事件类型
  TDropFileListBox = class(TListBox)           //从 TListBox 继承新的类
private
  FEnabled: Boolean;                          //属性 DropEnabled 的内部变量
protected
  FDropFile: TMyNotifyEvent;                 //事件指针
  procedure DropFiles(var Mes: TMessage); message WM_DROPFILES;
  procedure FDropEnabled(Enabled: Boolean);    //设置 DropEnabled 属性的过程
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
published
  property OnDropFiles: TMyNotifyEvent read FDropFile write FDropFile;
  property DropEnabled: Boolean read FEnabled write FDropEnabled;
end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('ZYKJ', [TDropFileListBox]); //注册组件到组件板上
end;
constructor TDropFileListBox.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FEnabled := true; //类被构造时，使 DropEnabled 的默认值为 True
end;
destructor TDropFileListBox.Destroy;
begin
  inherited Destroy;
end;
//改变属性 DropEnabled 的调用过程
```

```

procedure TDropFileListBox.FDrop Enabled(Enabled:Boolean);
begin
    FEnabled:=Enabled;
    DragAcceptFiles(Self.Handle,Enabled);    //设置组件窗口是否接受文件拖放
end;
//接受 WM_DropFiles 消息的过程
procedure TDropFileListBox.DropFiles(var Mes:TMessage);
var FN:TStringList;
    FileName:array [1..256] of char;
    sFN:String;
    i,Count,p:integer;
begin
    FN:=TStringList.Create;
    Count:=DragQueryFile(Mes.WParam,$FFFFFFFF,@FileName,256);
                                                    //得到拖放文件的个数

    For i:=0 to Count-1 do
        begin
            DragQueryFile(mes.WParam,i,@FileName,256);    //查询文件名称
            sFN:=FileName;
            p:=pos(chr(0),sFN);    //去掉文件名末尾的 ASCII 码为 0 的字符
            sFN:=copy(sFN,1,p-1);
            FN.Add(sFN);
        end;
    DragFinish(mes.WParam);    //释放所使用的资源
    if Assigned(FDropFile) then
        FDropFile(self, FN);    //调用事件，并返回文件名列表参数
    FN.Free;
end;
end.

```

【组件验证】

组件安装后，创建一个项目，把该组件加载到窗体中，可以发现该组件有一个 DropFiles 事件，编写该事件的程序代码如下：

```

procedure TForm1.DropFileListBox1DropFiles(Sender: TObject;
    FileNames: TStringList);
begin
    DropFileListBox1.Items.AddStrings(FileNames);
end;

```

程序的设计界面如图 11-14 所示，执行程序，从磁盘上拖动几个文件到列表框中后，程序的运行界面如图 11-15 所示。

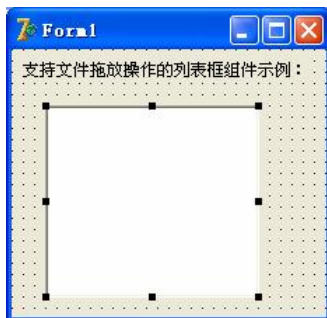


图 11-14 程序设计界面

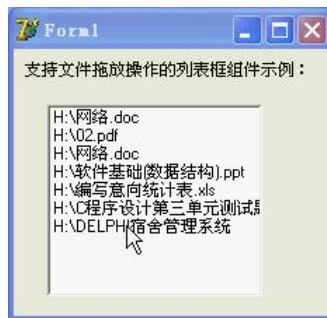


图 11-15 程序运行界面

11.3 上机练习

【练习题目】：连续工作的按钮组件

编写一个能够连续工作的按钮组件，当它被按下的时候，它所执行的功能就持续执行（也就是每隔一个固定的时间段执行一次按钮功能），当松开时，就停止其功能的执行。

【要点提示】

（1）该组件可结合两个组件的功能来实现：一个组件为按钮组件，一个组件为计时器组件。总体功能的实现思路如下：把按钮要实现的具体功能编写在计时器组件的 **Timer** 事件中，当按下按钮时，启动计时器工作，这样按钮要实现的功能就每隔一定的时间间隔实现一次，就好像按钮在连续工作；当从按钮上释放鼠标时，按钮将不再工作，可通过禁止计时器工作来实现。

（2）编写的组件应该具有以下功能：组件能够响应鼠标的按下和弹起事件；应该提供一个事件，在该事件中能够编写实现按钮功能的程序代码，该事件应在鼠标按下时每隔一定的时间间隔发生一次；为了控制按钮按下时，按钮功能程序代码执行的速度，可提供一个属性用来设置两次按钮功能代码执行的时间间隔。

【参考代码】

```
unit ButtonTimer;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, stdCtrls, Controls, Forms, Dialogs,
    ExtCtrls;
type
    TButtonTimer = class(TButton)
    private
        FTimer: TTimer;           //增加一个计时器组件作为成员
        FInterval: Integer;       //设置发生 OnPush 事件的时间间隔
        FOnPush: TNotifyEvent;
    protected
        procedure TimerTrigger(Sender: TObject);
```

```

Procedure SetInterval(Value :integer);           //设置时间间隔属性过程

procedure MyMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);         //按下鼠标过程
procedure MyMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);         //弹起鼠标过程

public
    Constructor Create(AOwner:TComponent); override;      //构造函数
    Destructor Destroy; override;                        //析构函数
published
    //声明 Interval 属性
    property Interval :Integer read FInterval write SetInterval default 200;
    //声明事件
    property OnPush: TNotifyEvent read FOnPush write FOnPush;
end;
procedure Register;
implementation
procedure Register;                                //注册
begin
    RegisterComponents('Samples', [TButtonTimer]);
end;
Constructor TButtonTimer.Create(AOwner: TComponent);
begin
    Inherited Create(AOwner);
    FTimer:=TTimer.Create(self);
    with FTimer do begin
        Ontimer:=TimerTrigger;           // TimerTrigger 过程作为计时器的 Timer 事件过程
        Enabled:=false;
    end;
    FInterval := 200;                      //属性 FInterval 的值为 200
    FTimer.Interval := 200;                //时钟成员的 Interval 的值为 200
    OnMouseDown:= MyMouseDown;
    // MyMouseDown 过程作为 OnMouseDown 的事件过程
    OnMouseUp := MyMouseUp;               // MyMouseUp 过程作为 OnMouseUp 的事件过程
end;
Destructor TButtonTimer.Destroy;          //析构函数
begin
    FTimer.free;
    inherited Destroy;
end;
procedure TButtonTimer.SetInterval(Value :integer);    //设置属性 Finterval 过程
begin
    FInterval:=value;
    FTimer.Interval:=value;
end;
//该过程作为 Timer 成员的 Timer 事件过程
procedure TButtonTimer.TimerTrigger(Sender: TObject);

```



```

begin
    if assigned(FOnPush) then          //在该过程中激发 OnPush 事件
        FOnPush(self);
end;
procedure TButtonTimer.MyMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    FTimer.enabled:=true;
end;
procedure TButtonTimer.MyMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
    FTimer.enabled:=False;
end;
end.

```

【组件测试】

编译并注册组件，新建一个新的项目，在窗体上添加一个标签、一个编辑框和一个本题创建的 TButtonTimer 组件。程序的设计界面如图 11-16 所示。设置 ButtonTimer 组件的 Caption 属性值为“数字增1”，Interval 属性值为 200。编写的 OnPush 事件代码如下：

```

procedure TForm1.ButtonTimer1Push(Sender: TObject);
begin
    Edit1.Text :=inttostr(strtoint(Edit1.text)+1);
end;

```

执行程序，按住 ButtonTimer1 组件，会发现编辑框中的数字不停地加 1（每隔 0.1 秒加 1），程序的运行界面如图 11-17 所示。



图 11-16 程序设计界面



图 11-17 程序运行界面

课后考场

一、选择题（20 分，每题 5 分）

- 能够在对象观察器中编辑的属性，应在类的_____部分声明。

A. Private	B. Public
C. Protected	D. Published

2. _____类型属性通常不能在类的 Published 部分定义，不能在对象观察器中编辑，只能在运行时使用。
- A. 基本 B. 枚举 C. 集合 D. 数组
3. _____类型属性在对象观察器中有一个“+”号，展开后有一系列的子项，子项的值有两个情况：True 和 False。
- A. 基本 B. 枚举 C. 集合 D. 数组
4. _____不是 TWinControl 类的特点。
- A. 有窗口句柄 B. 能够接受输入焦点
C. 能够作为其他组件的父组件 D. 占用很少的系统资源

二、填空题（40 分，每空 5 分）

1. 若类具有对象属性，一般需在类的_____函数中创建对象，在_____函数中释放对象。
2. _____类型的属性在对象观察器中显示为一个列表，用户可以从列表中选择属性值。
3. 创建组件的第一步通常是执行【Component】→【_____】菜单命令，以弹出【新建组件】对话框。
4. _____是通知事件类型，是大多数事件的类型，只有一个 Sender 参数。
5. 创建的组件图标的扩展名应为_____，其大小应为_____。
6. 为编译安装组件，用户可以新建一个包工程，包工程的扩展名为_____。

三、程序设计题（40 分）

编写一个显示当前时间的时钟组件，该组件每隔 0.5 秒显示一次当前时间；然后编写一个验证程序进行验证。

第 12 章 图形图像编程

本章要点

- ▮ TCanvas 对象的常用属性、方法及其使用
 - ▮ TGraphic 对象的常用属性、方法及其使用
 - ▮ TBitmap 对象的常用属性、方法及其使用
 - ▮ TPicture 对象的常用属性、方法及其使用
 - ▮ Delphi 中的图形图像组件及其使用
-

12.1 理论知识

12.1.1 TCanvas 对象的使用

在 Delphi 中, TCanvas 对象不是一个组件, 而是窗体、图像框、PaintBox 等组件的一个对象属性。该对象相当于一块画布, 使用它的属性和方法, 可以很灵活地绘制出直线、椭圆、矩形等图形。

1. TCanvas 对象的属性

TCanvas 对象的基本属性有 Pen (画笔)、Brush (画刷) 和 Pixels (像素), 以及一些其他属性。

(1) Pen 属性

该属性也是一个对象, 相当于绘图时的“画笔”。Pen 属性也有自己的属性, 通过设置它的属性值可以改变画图时的线条宽度、样式和颜色等。下面介绍 Pen 属性的常用属性。

① Color 属性: 用于设置所画线条的颜色, 默认时为黑色。该属性的设置方法有三种, 分别如下:

- ▮ 使用 Delphi 提供的系统颜色常量, 如语句:

```
Form1.Canvas.Pen.Color:=clBlue
```

该语句的作用是把 Form1 的 Canvas 对象的画笔颜色设置为蓝色。表 12-1 列出了 Delphi 系统的颜色常量及其表示的色彩。

- ▮ 使用 RGB 函数, RGB 函数的一般格式如下:

```
RGB(R,G,B)
```

参数 R、G、B 分别代表红、绿、蓝三种颜色的成分, 取值在 0~255 之间。如语句:

```
Form1.Canvas.Pen.Color:=RGB(0,0,255)
```

其作用与语句“Form1.Canvas.Pen.Color:=clBlue”的作用是一样的。

表 12-1 颜色常量及其表示的颜色

颜色常量	表示的颜色	颜色常量	表示的颜色
clAqua	浅绿色	clMenu	当前菜单的背景颜色
clBlack	黑色	clWindow	当前窗体的背景色
clBlue	蓝色	clWindowFrame	当前窗体框架的颜色
clDkGray	深灰（灰黑色）	clMenuText	当前菜单文字的颜色
clFuchsia	紫红色	clWindowText	当前窗体中文字的颜色
clGray	灰色	clCaptionText	当前标题的颜色
clGreen	绿色	clActiveBorder	当前激活的边界的颜色
clLime	橙色	clInactiveBorder	当前没被激活的边界的颜色
clLtGray	浅灰色	clAppWorkSpace	当前工作区的颜色
clMaroon	枣栗色	clHighlight	当前的高亮色
clNavy	深蓝色	clHightlightText	当前的高亮文本的颜色
clOlive	橄榄绿	clBtnFace	当前按钮的颜色
clPurple	紫色	clBtnShadow	当前按钮背影的颜色
clRed	红色	clGrayText	当前灰色字体的颜色
clSilver	银灰色	clInactiveCaptionText	当前没被激活的窗体标题文字的颜色
clTeal	Teal 青灰色（一种蓝）	clBtnHighlight	当前按钮高亮的颜色
clWhite	白色	cl3DDkShadow	当前 3D 按钮的颜色（只用于 Windows 95 或 NT 4.0）
clYellow	黄色	cl3DLight	高亮的 3D 按钮的颜色（只用于 Windows 95 或 NT 4.0）
clBackground	当前桌面的背景色	clInfoText	工具栏提示字符的颜色（只用于 Windows 95 或 NT 4.0）
clActiveCaption	当前窗体的标题栏色	clInfoBk	工具栏提示背景色（只用于 Windows 95 或 NT 4.0）
clInactiveCaption	当前不被激活的标题栏的颜色		

1 直接以数字表示颜色。在 Delphi 7 中，可通过在十六进制数前面加\$符号来表示颜色。因为 Windows 98 中的颜色系统是 32 位的，故其前面的 8 位不使用，一般写成 0。后面的每 8 位表示一种颜色分量，从低到高依次是红、绿、蓝分量的取值。例如，如下语句：

```
Form1.Canvas.Pen.Color:=$00FF00FF;
```

该语句与下面的语句作用是相同的：

```
Form1.Canvas.Pen.Color:=RGB(255,0,255);
```

提示：4 位二进制数可用 1 位十六进制数表示。

② Style 属性：用于设置线条的样式，如：实心线、虚线、点划线等。各种线条样式的含义如表 12-2 所示。该属性的默认值为 PsSolid（实线）。

表 12-2 Style 属性值及说明

线条样式	含 义
PSClear	绘制的线条不可见
PSSolid	绘制实线
PSDash	绘制由虚线
PSDot	绘制圆点组成的线条
PSDashDot	绘制点划线
PSDashDotDot	绘制双点划线

例如，下面语句的作用是把 Form1 窗体的 Canvas 的画笔风格设置为点划线：

```
Form1.Canvas.Pen.Style:= PSDashDot;
```

③ **Width** 属性：用于设置线条的粗细宽度，单位是像素，其最小值为 1。下面的代码会将线条的粗细宽度设置为 10 个像素。

```
Form1.Canvas.Pen.Width:= 10;
```

④ **Mode** 属性：用于设置画笔的颜色与背景颜色之间的作用效果，效果方案也有很多。例如，一直为黑色、一直为白色、将屏幕的颜色反转等。**Mode** 属性的取值及其含义如表 12-3 所示。

表 12-3 Mode 属性的取值及其含义

取 值	含 义
pmBlack	一直黑色
pmWhite	一直白色
pmNop	不改变，所有的图像都不画，使绘图失效
pmNot	将屏幕上的颜色反转
pmCopy	以 Color 属性设置颜色
pmNotCopy	将画笔的颜色反转
pmMask	将画笔的颜色与屏幕的颜色进行“与”操作
pmMaskPenNot	将画笔的颜色与反转后的屏幕的颜色进行“与”操作
pmMaskNotPen	将屏幕的颜色与反转后的画笔的颜色进行“与”操作
pmMerge	将画笔的颜色与屏幕的颜色进行“或”操作
pmMergeNotPen	将屏幕的颜色与反转后的画笔的颜色进行“或”操作
pmMergePenNot	将画笔的颜色与反转后的屏幕的颜色进行“或”操作
pmNotMerge	将画笔的颜色与屏幕的颜色进行“或”操作，再把结果按位取反
pmNotMask	将画笔的颜色与屏幕的颜色进行“与”操作
pmXor	将画笔的颜色与屏幕的颜色进行“异或”操作
pmNotXor	将画笔的颜色与屏幕的颜色进行“异或”操作，再把结果按位取反

例如，有如下语句：

```
Form1.Canvas.Pen.Mode:=pmXor;
```

其作用是画线时把画笔的颜色与屏幕的颜色进行异或操作。

(2) Brush 属性

Brush 属性也是一个对象属性，相当于绘图时的画刷。使用该属性可设置在图形内或画布上某区域的填充方式，同样它也有自己的属性。可以通过它的属性来设定填充颜色、图案等。下面介绍 Brush 对象属性的常用属性。

① Color 属性：用于设置填充色，取值及含义与 Pen 的 Color 属性一样。

② Style 属性：用于设置填充样式。图 12-1 所示为 Brush 颜色值为 clBlack 时，Style 属性值及其代表的样式。

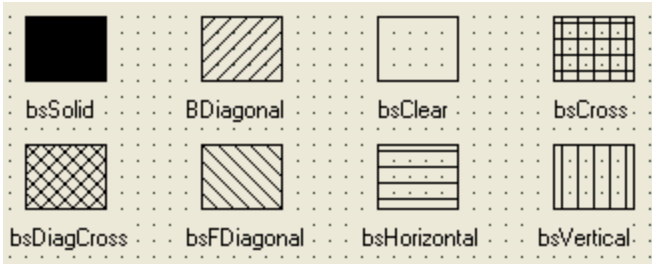


图 12-1 Brush 对象的 Style 属性值及其代表的填充样式

(3) Pixels 属性

该属性是一个二维数组属性，其声明如下：

```
property Pixels[X,Y:Integer]:TColor;
```

该属性是一个二维数组属性，数组中的每个元素用来存放画布中一个像素所对应的颜色值。在画布上绘图，实际上就是改变画布上某些点所对应的像素值。一般只有在必须访问单个像素颜色时，才使用 Pixels 数值。属性中的 X、Y 表示像素的坐标，坐标的原点在 Canvas 对象的左上角。

(4) Font 属性

该属性用于设置在画布对象上输出文字的字体格式，它也有自己的属性，通过设置它的属性，可以决定在画布上输出文字的字体、大小、颜色及字体风格等。

(5) ClipRect 属性

该属性的声明如下：

```
property ClipRect:TRect;
```

该属性用于对图形进行剪裁，使它只显示在指定的矩形区域内。当输出图形大于给定值时，超出的部分将被裁剪。

(6) CopyMode 属性

该属性用于设置从其他画布拷贝图像的方式，其取值及含义如表 12-4 所示，该属性的默认值为 cmSrcCopy（表示拷贝的像素将覆盖画布上原有的像素）。

表 12-4 CopyMode 的属性值及其含义

CopyMode 属性的取值	含 义
cmSrcCopy	拷贝的像素将覆盖画布上原有的像素
cmBlackness	以黑色输出
cmDsInvert	将目标图像反转
cmMergeCopy	将原先的图像和源图像进行“与”操作
cmMergePaint	将源图像反转后，再和目标图像进行“或”操作
cmNotSrcCopy	将源图像反转拷贝
cmNotSrcErase	先将源图像和目标图像进行“或”操作，再将结果反转
cmPatCopy	将源图像进行“异或”操作后，再拷贝到目标设备
cmPatInvert	将目标点位图与图案进行“异或”操作
cmPatPaint	先将源点位图反转后，再和图案进行“或”操作
cmSrcAnd	将目标像素与源点位图进行“与”操作
cmSrcCopy	拷贝并覆盖目标点位图
cmSrcErase	先将目标点位图反转，再与源点位图进行“与”操作
cmSrcInvert	将源图和目标图的像素进行“异或”操作
cmSrcPaint	将源图和目标图的像素进行“或”操作
cmWhiteness	将所有的输出变成白色

(7) Handle 属性

该属性用于返回画布对象的句柄。

(8) PenPos 属性

该属性的类型是 Tpoint 型，用于设置或返回当前画笔位置所在的点。

2. TCanvas 对象的方法

利用 Tcanvas 对象的方法可以完成常用的绘图功能，如绘制直线、椭圆、矩形等。TCanvas 对象最常用的方法如下。

(1) MoveTo 方法

[格式]:

```
procedure MoveTo(X,Y Integer);
```

[功能]: 将画笔移到坐标 (X, Y) 指定的位置，使该点成为当前位置。

(2) LineTo 方法

[格式]:

```
procedure LineTo(X,Y Integer);
```

[功能]: 使用画笔从当前位置画一条直线到指定位置 (X, Y)，画好后，画笔的位置将移至坐标 (X, Y) 处。

(3) Arc 方法

[格式]:

```
procedure Arc(X1,Y1,X2,Y2,X3,Y3,X4,Y4: Integer);
```

[功能]: 用于画一段椭圆弧。其中, 椭圆是由点 (X1,Y1) 和点 (X2,Y2) 所确定的矩形所决定, 弧的起点是椭圆中心和点 (X3,Y3) 的连线与椭圆的交点, 弧的终点是椭圆中心和点 (X4,Y4) 的连线与椭圆的交点, 并以逆时针方向画弧。

(4) Ellipse 方法

[格式]:

```
procedure Ellipse(X1,Y1,X2,Y2: Integer);
```

[功能]: 用于绘制椭圆, 该椭圆是由点 (X1,Y1) 和点 (X2,Y2) 所决定矩形的内切椭圆。椭圆的边界使用当前的 Pen (画笔) 绘制, 椭圆的内部使用当前的 Brush (画刷) 填充。

(5) Rectangle 方法

[格式]:

```
procedure Rectangle(X1,Y1,X2,Y2: Integer);
```

[功能]: 用于绘制矩形, 该矩形是由点 (X1,Y1) 和点 (X2,Y2) 所确定。矩形的边界使用当前的 Pen 绘制, 矩形的内部使用当前的 Brush 填充。

(6) RoundRect 方法

[格式]:

```
procedure RoundRect(X1,Y1,X2,Y2,X3,Y3: Integer);
```

[功能]: 用于绘制带有圆角的矩形, 该圆角矩形的矩形范围是由点 (X1,Y1) 和点 (X2,Y2) 所确定, X3、Y3 分别指定圆角的长半度和短半径。

(7) Pie 方法

[格式]:

```
procedure Pie(X1,Y1,X2,Y2,X3,Y3,X4,Y4: Integer);
```

[功能]: 用于绘制扇形 (椭圆的一部分)。该椭圆内切于由点 (X1,Y1) 和点 (X2,Y2) 所确定的矩形, 扇形区域是由椭圆中心到 (X3,Y3) 和 (X4,Y4) 两点所组成的椭圆部分所决定。并且扇形的内部使用当前的 Brush 填充。

(8) Draw 方法

[格式]:

```
procedure Draw(X,Y Integer;Graphic:TGraphic);
```

[功能]: 用于在画布上由 (X, Y) 指定的位置处, 输出由 Graphic 参数所指供的图像, 该图像可以是位图, 也可以是图标或图元文件。

注意: 要想绘制已存在的图像文件, 可先建立一个图形对象, 使用它将图像从文件中读出来, 然后再以它为参数, 调用 Draw 方法将图像画到画布上去。

(9) CopyRect 方法

[格式]:

```
procedure CopyRect(Dest:TRect;Canvas:TCanvas;Source:TRect);
```

[功能]: 用于将一个画布上的一部分图形图像拷贝到当前画布上。其中, Dest 参数指定

目标画布的区域, **Canvas** 参数指定源画布, **Source** 参数指定源画布上要拷贝的矩形区域。

(10) **FillRect** 方法

[格式]:

```
procedure FillRect(const Rect:TRect);
```

[功能]: 使用当前的 **Brush** 填充由参数 **Rect** 所指定的矩形区域。

(11) **StretchDraw** 方法

[格式]:

```
procedure StretchDraw(const Rect:TRect; Graphic:TGraphic);
```

[功能]: 用于将 **Graphic** 参数指定的图像输出在 **Rect** 参数指定的区域内, 并将图像自动放大或缩小以适应区域。

(12) **TextOut** 方法

[格式]:

```
procedure Textout(X,Y:Integer, const Text:String);
```

[功能]: 用于在画布上以当前字体输出字符串。输出的字符串由参数 **Text** 指定, 输出的位置由参数 **X** 和 **Y** 指定。

(13) **Polygon** 方法

[格式]:

```
procedure Polygon([Point(X1,Y1),Point(X2,Y2),Point(X3,Y3),...]);
```

[功能]: **Polygon** 方法是用于画由一系列点定义的多边形, 其中 (**Xi**, **Yi**) 为多边形的顶点。绘制的多边形为封闭图形。

(14) **Chord** 方法

[格式]:

```
procedure Chord(X1,Y1,X2,Y2,X3,Y3,X4,Y4);
```

[功能]: **Chord** 方法用来连接椭圆上的两点, 以形成“帽图”。椭圆由 (**X1**, **Y1**)、(**X2**, **Y2**) 两点所确定的矩形决定, 连线的起点为 (**X3**, **Y3**), 连线的终点是 (**X4**, **Y4**)。

【例 12-1】 编写一个程序, 用来绘制一个椭圆和几个圆角矩形。程序执行时, 单击【画图】按钮, 运行界面如图 12-2 所示。

【实现分析】

画椭圆可使用 **TCanvas** 对象的 **Ellipse** 方法, 画圆角矩形可使用 **TCanvas** 对象的 **RoundRect** 方法。在画椭圆时, 给它填充的是一幅位图, 因此应创建一个位图对象并赋值给 **Brush** 属性的 **Bitmap** 属性。画的三个圆角矩形的填充风格均不同, 可通过设置 **TCanvas** 的 **Brush.Style** 属性来设置填充风格。另外填充色也不同, 可通过设置 **TCanvas** 的 **Brush.Color** 属性来设置填充色。

【界面设计】

为窗体添加一个 **Button** 组件, 设置它的 **Caption** 属性值为“画图”。

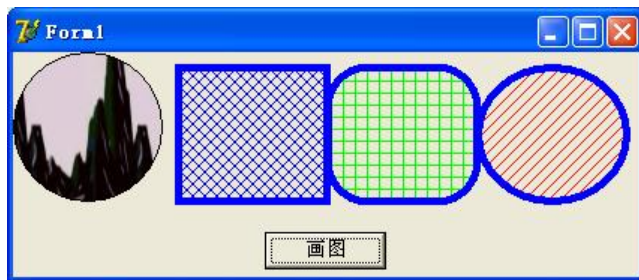


图 12-2 程序运行界面

【程序代码】

主要的程序代码如下：

```

procedure TForm1.Button1Click(Sender: TObject);
var
    bitmap:TBitmap;           //用来存放创建位图，该位图作为填充图像
    CurDir:String;            //存放应用程序当前目录
begin
    GetDir(0,CurDir);         //获得应用程序的当前目录
    With Canvas do
    begin
        Bitmap:=TBitmap.Create; //创建位图
        try
            Bitmap.LoadFromFile(CurDir+'\1.BMP'); //装载位图
            Brush.Bitmap:=Bitmap; //把创建的位图赋值给 Brush.Bitmap，作为填充图像
            Ellipse(0,0,100,100); //画椭圆，以位图填充
        finally
            Form1.Canvas.Brush.Bitmap:=nil; //卸除 Bitmap 位图
        end;
    end;
    With Canvas do
    begin
        pen.Color:=RGB(0,0,255); //设置画笔颜色为蓝色
        pen.Width:=5; //设置画笔的宽度为 5 个像素
        brush.Color:=$00FF0000; //设置画刷的填充颜色为蓝色
        brush.Style:=bsDiagCross; //设置画刷的填充风格为交叉对角线
        RoundRect(110,10,210,100,0,0); //画圆角矩形
        brush.Color:=$0000FF00; //设置画刷的填充颜色为绿色
        brush.Style:=bsCross; //设置画刷的填充风格为交叉线
        RoundRect(210,10,310,100,50,50); //画圆角矩形
        brush.Color:=$000000FF; //设置画刷的填充颜色为红色
        brush.Style:=bsBDiagonal; //设置画刷的填充风格为斜对角线
        RoundRect(310,10,410,100,100,100); //画圆角矩形
    end;
end;

```

12.1.2 TGraphic 对象的使用

在 TCanvas 对象的 Draw 和 StretchDraw 方法中将使用 TGraphic 型的参数。TGraphic 对象是 TBitmap、TIcon、TMetafile 对象的抽象基类，如果不知道图像的具体类型，可以使用它来存放。

1. TGraphic 对象的属性

TGraphic 对象有 Height、Width、Empty 等属性。Height 和 Width 属性用来设置对象的宽度和高度。Empty 属性是一个 Boolean 型的只读属性，其值为 True 时，表示 TGraphic 对象中已经包含有图像，其值为 False 时，表示 TGraphic 对象中不包含图像。

2. TGraphic 对象的方法

TGraphic 对象的方法如下。

(1) LoadFromFile 方法

[格式]:

```
procedure LoadFromFile(const FileName: String);
```

[功能]: 该方法从参数 FileName 作为文件名指定的文件中读取图像，并装入 TGraphic 对象中。

(2) SaveToFile 方法

[格式]:

```
procedure SaveToFile(const FileName: String);
```

[功能]: 该方法把 TGraphic 中的图像保存到指定的文件中，保存文件的文件名由参数 FileName 指定。

12.1.3 TPicture 对象的使用

TPicture 对象可以存储任意类型的图像，它是 TGraphic 对象的容器。

1. TPicture 对象的属性

TPicture 对象的属性如下。

(1) Bitmap 属性

该属性的类型为 TBitmap 型，若 Tpicture 对象包含的是位图，可使用该属性指定位图。

(2) Icon 属性

该属性的类型为 TIcon 型，若 Tpicture 对象包含的是图标，可使用该属性指定图标。

(3) Metafile 属性

该属性的类型为 TMetafile 型，若 TPicture 对象包含的是图元文件，可使用该属性指定图元文件。

(4) Graphic 属性

该属性用于设置或返回 TPicture 对象中包含的图像。

2. TPicture 对象的方法

TPicture 对象的主要方法有 LoadFromFile 和 SaveToFile，这两个方法的格式和功能与

TGraphic 对象的同名方法完全一致。

【例 12-2】 编写一个绘制位图的程序，程序的运行界面如图 12-3 所示。程序执行时单击【绘制位图】按钮，将从窗体的左上角开始原样绘制一幅图片，然后再把该图片缩小在一个矩形框内绘制，如图 12-3 所示。



图 12-3 程序运行界面

【实现分析】

绘制位图可以使用 Canvas 对象的 Draw 方法和 StretchDraw 方法，Draw 方法用于原样绘制位图，StrtechDraw 方法用于把一个位图在一个矩形框中绘制，位图自动调整大小以填满矩形框。

【界面设计】

为窗体添加一个 Button 组件，设置它的 Caption 属性值为“绘制位图”。

【程序代码】

主要的程序代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
var
    CurDir:String;
    Picture:Tpicture;
begin
    GetDir(0,CurDir);
    Picture:=Tpicture.create;
    Picture.LoadFromFile(CurDir+'1.bmp');      //从文件中装载位图
    With Canvas do
        begin
            Draw(0,0,Picture.Graphic );      //从画布的左上角原样绘图
            StretchDraw(Rect(0,0,150,150),picture.Graphic );
            //在一个矩形区域内绘图，图自动变化大小以适应矩形框
        end
    end;
end;
```

12.1.4 TBitmap 对象的使用

TBitmap 对象可以包含一个位图图像, 拥有位图图像和调色板的句柄, 可以自动管理调色板。另外 TBitmap 对象还具有 Canvas 属性, 利用它可以绘制图形。

1. TBitmap 对象的属性

TBitmap 对象的主要属性如下。

(1) Handle 属性

该属性的类型是 HBitmap, 用于返回位图图像的句柄。在调用 Windows API 时, 许多 API 函数均需要传递图像的句柄, 通过该属性就可以获得 TBitmap 对象中的位图图像的句柄。

(2) IgnorePalette 属性

该属性是一个布尔型属性, 若其值设置为 True, 则表示 TBitmap 对象不使用调色板。

(3) Monochrome 属性

该属性是一个布尔型属性, 若其值设置为 True 时, 则表示 TBitmap 对象将显示为单色(黑白图像); 若其值设置为 False 时, 则表示 TBitmap 对象将显示为彩色。

(4) Palette 属性

该属性的类型是 HPalette 型, 表示的是 TBitmap 对象的调色板。若 Palette=0, 则表示对象不使用调色板, 此时可以使用 CreatePalette 方法来创建自己的调色板。

2. TBitmap 对象的方法

TBitmap 对象的主要方法有 LoadFromFile、SaveToFile、Releasehandle 等。LoadFromFile 方法和 SaveToFile 方法的格式和功能与 TGraphic 对象的同名方法完全一致。

Releasehandle 方法的格式与功能如下。

[格式]:

```
Function Releasehandle:Hbitmap;
```

[功能]: 释放 TBitmap 对象的调色板。

12.1.5 Delphi 中的图形图像组件

1. TShape 组件

TShape 组件的作用是绘制图形, 它的主要属性有 Brush、Pen、Shape 等。Brush 和 Pen 属性与 TCanvas 对象的相应属性基本相同, 此处不再赘述。Shape 属性用来设置 TShape 组件显示的图形形状, 其取值及其表示的图形如表 12-5 所示。

表 12-5 Shape 属性

属 性 值	图 形	属 性 值	图 形
stCircle	圆	stEllipse	椭圆
stRectangle	矩形	stRoundRect	圆角矩形
stSquare	正方形	stRoundSquare	圆角正方形

【例 12-3】 编写一个 Shape 形状演示程序, 程序的设计界面如图 12-4 所示。程序运行

时，单击【随机画图】按钮，TShape 组件的形状将依次在 6 种图形之间转换，同时在 Label 组件中显示图形的名称，TShape 组件的填充风格也自动在 8 种填充风格之间转换，并且填充的色彩随机产生。每种图形的转换时间为 200 毫秒。程序的运行界面如图 12-5 所示。

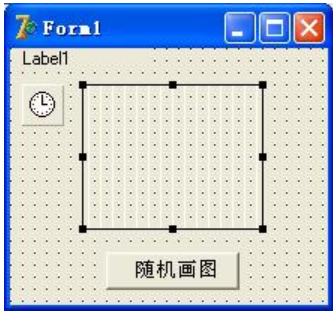


图 12-4 程序设计界面



图 12-5 程序运行界面

【实现分析】

为了能够使图形每隔 200 毫秒转换一次，可使用一个 TTimer 组件，把该组件的 Interval 属性设置为 200，在该组件的 Timer 事件中改变图形。为使图形依次在 6 种图形之间转换，可设一个单元（窗体/模块）级变量 Num，其初值为 0，在 Timer 事件中用该变量的值除以 6 求余，显然余数在 0~5 之间，设定每一个数对应一种图形，通过 Case 多分支语句进行判断并给 TShape 组件的 Shape 属性赋相应的值。填充风格也同样设置，只是应除 8 取余。填充色是随机的，可通过 Random 函数产生 3 个 255 之内的随机整数作为红、绿、蓝三色的分量，通过 RGB 函数产生色彩并赋值给 Brush 的 Color 属性。在 Timer 事件的最后使变量 Num 的值加 1。

【界面设计】

本例组件对象及其属性设置如表 12-6 所示。

表 12-6 例 12-3 组件对象及其属性设置

对 象 名	属 性 名	属 性 值	说 明
Label1			显示图形形状名
Shape1			顺序显示图形形状
Timer1	Interval Enabled	200 False	每隔 200 毫秒设置一次图形形状、图形的填充风格和填充色
Button1	Caption	'随机画图'	单击它进行初始化工作并启动计时器组件工作

【程序代码】

```
implementation
var
    Num:Integer; //Num 表示Timer1 的Timer 事件的发生次数
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```

Num:=0; //Num 的初值为 0
Timer1.Enabled :=True; //启动 Timer1, 开始计时
Randomize; //随机数初始化
end;
procedure TForm1.Timer1Timer(Sender: TObject);
var
    k,r,g,b:Integer;
begin
    k:=Num Mod 6; //k 为 6 种形状之一
    Case k of //本 Case 语句根据 k 的值决定显示何种图形
        0: begin Shape1.Shape:=stCircle; Label1.caption:='圆形';end;
        1: begin Shape1.Shape :=stEllipse; Label1.caption:='椭圆';end;
        2: begin Shape1.Shape :=stRectangle; Label1.caption:='矩形';end;
        3: begin Shape1.Shape :=stRoundRect; Label1.caption:='圆角矩形';end;
        4: begin Shape1.Shape :=stSquare; Label1.caption:='正方形';end;
        5: begin Shape1.Shape :=stRoundSquare;Label1.caption:='圆角正方形';end;
    end;
    k:=Num Mod 8; //此时 K 代表填充图案, 共有 8 种
    Case k of //本 case 语句根据 k 的值决定以何种图案进行填充
        0:Shape1.Brush.Style :=bsBDiagonal;
        1:Shape1.Brush.Style :=bsClear;
        2:Shape1.Brush.Style :=bsCross;
        3:Shape1.Brush.Style :=bsDiagCross;
        4:Shape1.Brush.Style := bsFDiagonal;
        5:Shape1.Brush.Style :=bsHorizontal;
        6:Shape1.Brush.Style := bsSolid;
        7:Shape1.Brush.Style :=bsVertical;
    end;
    Num:=Num+1; //Timer1 的 Timer 事件又执行了一次
    r:=random(255);g:=random(255);b:=random(255); //随机产生红、绿、蓝三种色彩成分
    Shape1.Brush.Color :=RGB(r,g,b); //作为填充色
    if Num>=1000 then //只执行 1000 次
        Application.Terminate ;
    end;
end;

```

2. TImage 组件

TImage 组件主要用于显示和处理图像, 能够显示 BMP、JPEG、ICO、WMF、EMF 等多种格式的图片。它的主要属性如下。

(1) Canvas 属性

画布属性, 同 Tcanvas 对象。

(2) Picture 属性

该属性用于指定在 Image 组件中显示的图像。可在设计阶段在属性窗口 (对象浏览器) 中找到该属性, 单击属性后面的【...】按钮, 以选择一幅图片在 TImage 组件中显示。也可以在运行时通过函数 LoadFromFile 为 TImage 组件加载一幅图片。例如, 有如下语句:

```
Image1.Picture.LoadFromFile('D:\A1.JPG');
```

该语句的作用是把“D:\A1.JPG”图像文件显示在 TImage 组件中。

(3) Stretch 属性

该属性是一个 Boolean 型属性, 其值为 True 时, 表示将自动对图像进行拉伸或缩小操作, 调整图像的尺寸以填满整个 TImage 组件; 当其值为 False 时, 若图像比 TImage 组件大, 多余的部分将显示不出来, 若图像比 TImage 组件小, 则图像没有填满 TImage 组件。

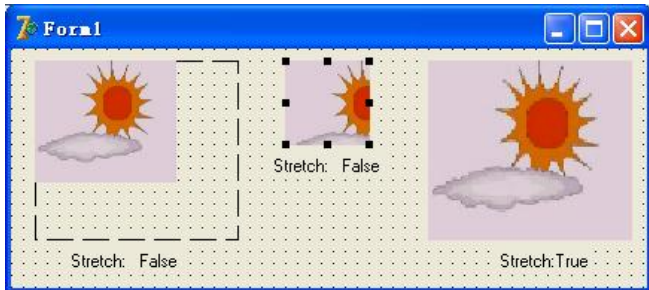


图 12-6 TImage 组件的 Stretch 属性取值及图像显示情况

(4) Center 属性

该属性是一个 Boolean 型属性, 用于确定图像是否居中显示在 TImage 组件中。若 Center 设置为 True, 则表示图像将居中显示; 否则, 图像将从 TImage 组件的左上角开始显示。

3. TPaintBox 组件

TPaintBox 组件主要提供一个可以用来绘制几何图形的矩形区域, 它有一个 Canvas 属性, 可以使用它的绘图方法来绘制图形。TPaintBox 组件的使用方法与 Form 窗体基本上一样, 不同的是在 TPaintBox 组件中绘图时, 绘图的坐标是以 TPaintBox 组件的左上角为原点, 而不是以窗体的左上角为原点。当 TPaintBox 组件的背景色与 Form 相同时, 在其上绘图和在窗体上绘图的效果一样。可见, 使用 TPaintBox 组件绘图的惟一好处就是将图形绘制在 TPaintBox 组件内, 省去了计算坐标的麻烦。其使用方法与窗体绘图基本一致, 此处不再赘述。

12.2 典型实例

12.2.1 典型实例一

【实例题目】：背景填充程序

编写一个以位图填充窗体背景的应用程序, 程序的设计界面如图 12-7 所示。程序运行时, 单击【选择图片】按钮, 将弹出【打开】对话框供用户选择一幅作为背景填充的图片, 并根据单选钮的状态进行“平铺”或“拉伸”填充。单击【平铺】单选钮, 图片将以“平铺”方式填充整个窗体, 如图 12-8 所示。单击【拉伸】单选钮, 图片将以“拉伸”方式填充整个窗体, 如图 12-9 所示。



图 12-7 程序设计界面



图 12-8 程序运行界面（一）



图 12-9 程序运行界面（二）

【实现方法】

为实现在矩形区域中填充位图，可先给 Brush 对象的 Bitmap 赋一个位图对象，然后再绘制出矩形，该矩形将以指定的位图进行填充。但该方法无法实现位图的拉伸填充，要实现位图的拉伸填充，可使用 TCanvas 对象的 StretchDraw 方法。该方法以某一个 TGraphic 对象中包含的图形在一个矩形区域内拉伸填充，因此使用该方法之前应产生一个 TGraphic 对象。

【界面设计】

本实例组件属性设置及组件作用如表 12-7 所示。

表 12-7 组件的属性设置及组件作用

对 象 名	属 性 名	设 置 值	对 象 作 用
Button1	Caption	'选择图片'	单击它出现打开对话框供用户选择填充图片
OpenDialog1	Filter	'位图文档*.bmpJPEG 文档*.jpg'	用来选择图片文件
RadioButton1	Caption	'平铺'	用来设置以“平铺”方式填充
RadioButton2	Caption	'拉伸'	用来设置以“拉伸”方式填充

【程序代码】

本例主要程序代码如下：

```
implementation
var
  FName:String;           //背景填充的图片文件名
{$R *.dfm}
procedure bTile();        //平铺填充
var
  Bitmap:TBitmap;
begin
  Bitmap:=TBitmap.Create; //创建位图对象
  Bitmap.LoadFromFile(FName); //装载图像
  Form1.Canvas.Brush.Bitmap :=Bitmap; //把位图对象赋值给 Brush.bitmap，以便填充
  Form1.Canvas.rectangle(0,0,Form1.width,Form1.height);
  //在整个窗体内画矩形，用位图填充,平铺效果
  Bitmap:=nil;           //卸载位图
end;
procedure bStretch();    //拉伸填充
var
```

```

    psicture:TPicture;
begin
    Picture:=TPicture.Create ;           //创建 Picture 位图
    Picture.LoadFromFile(FName );        //装载图像
    Form1.Canvas.Stretchdraw(Rect(0,0,Form1.width,Form1.height),Picture.Graphic );
                                           //使用 StretchDraw 方法在整个窗体上绘图，拉伸效果
    Picture:=nil;                        //卸载 Picture 对象
end;
procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:String;
begin
    GetDir(0,CurDir);                    //获取当前路径
    FName:=CurDir+'\Default.BMP';       //设置默认填充的图片名
end;
procedure TForm1.FormPaint(Sender: TObject);
begin
    if RadioButton1.Checked then
        bTile
    else
        bstretch;
end;
procedure TForm1.Button1Click(Sender: TObject);    //选择一个图片文件以填充
begin
    if OpenFileDialog1.Execute then                //弹出【打开】对话框以选择文件
    begin
        FName:=OpenDialog1.FileName ;
        if RadioButton1.Checked then                //如果选择了【平铺】单选钮
            btile                                    //“平铺”填充
        else
            bStretch;                                //“拉伸”填充
    end;
end;
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    BTile;                                           //平铺填充
end;
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    Bstretch;                                        //拉伸填充
end;

```

12.2.2 典型实例二

【实例题目】：照片自动展示程序

制作一个照片自动展示程序，程序启动后将循环显示照片，每张照片在屏幕上停留 0.5

秒。当用户单击图片时将显示一个提示框，询问用户是否停止展示，如果用户选择【是】将停止展示，选择【否】将接着展示。程序的设计界面如图 12-10 所示，程序的运行界面如图 12-11 所示。



图 12-10 程序设计界面



图 12-11 程序运行界面

【实现方法】

本实例要求每隔 0.5 秒展示一张照片，可用一个 `TTimer` 组件来实现。由于每次展示的照片并不是同一张照片，故在 `TTimer` 组件的 `Timer` 事件中还应形成要展示的照片文件名。照片文件命名必须符合一定的规则，本例规定照片文件名分别为 `TZY1.jpg`，`TZY2.JPG`，`TZY3.JPG`，……，直到 `TZJ08.JPG`。为形成文件名，可设一个单元（窗体/模块）级变量（假设为 `Num`），在每次发生 `Timer` 事件时，取 `Num` 除以 8 的余数，该余数加 1 作为要显示照片的照片序号，再根据它不难形成照片文件名。然后让 `Num` 的值加 1，准备显示下一张照片。

【界面设计】

在窗体上放置一个 `TImage` 组件（`Image1`），用来显示照片，设置它的 `align` 属性值为 `alClient`。再放置一个 `TTimer` 控件（`Timer1`），用来每隔 0.5 秒显示一张照片。

【程序代码】

本例的程序代码如下：

```
implementation
var
    Num:Integer;                //表示计时器控件的 Timer 事件发生的次数
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
    Num:=0;
    Image1.Stretch :=True;      //使照片充满整个图像框
    Timer1.Interval:=500;       //每隔 0.5 秒发生一次 Timer 事件
    Timer1.Enabled :=True;      //启动计时器控件
end;
procedure TForm1.Timer1Timer(Sender: TObject);
var
    xh:Integer;
    CurDir,PicFile:string;
begin
```

```
XH := Num Mod 8 ;           //得到要展示的照片文件的序号
GetDir(0, CurDir);
PicFile:= CurDir+'tzy'+inttostr(XH+1)+'.jpg';           //形成要展示文件的文件名
Image1.Picture.LoadFromFile (PicFile);                 //展示照片
Num := Num + 1;           //准备展示下一张照片
if Num=1000 Then           //如果 Num 的值很大，再把它置 0
    Num:=0;
end;
procedure TForm1.Image1Click(Sender: TObject);
begin
    if MessageDlg('退出照片展示吗? ',mtConfirmation, [mbYes, mbNo],0) = mrYes then
        begin
            close;
        end;
    end;
end;
```

程序的运行界面如图 12-11 所示。运行时如果单击照片，将会出现如图 12-12 所示的退出提示信息，在该提示框中单击【Yes】按钮将退出图片展示程序，单击【No】将继续展示。



图 12-12 提示信息

12.3 上机练习

12.3.1 上机练习一

【练习题目】：转动的秒针

编写一个程序，程序运行时将出现一个钟表的轮廓和一个秒针，如图 12-13 所示。秒针以每秒 12 度的速度移动，图 12-14 是某个时刻的程序运行情况。

【要点提示】

(1) 秒针移动和实现。秒针可通过画线来实现，为演示出秒针的移动，可画两次直线：一次用来覆盖掉原来的线条，方法是使背景色为白色，然后置画笔模式为 `pmNotXor`；另一次是计算出新的坐标位置，然后以 `pmCopy` 模式画线条。由于每隔一段时间就要移动指针，故可使用一个计时器组件，设置它的 `Interval` 属性值为 1000，使它每秒钟发生一次 `Timer` 事件，在 `Timer` 事件中实现覆盖原来指针和重画指针的功能。



图 12-13 程序设计界面



图 12-14 程序运行界面

(2)画线的线条终点的计算。假设指针在窗体的坐标位置为 (x_1, y_1) 至 (x_2, y_2) , 以 (x_1, y_1) 为圆心, 按顺时针方向改变 (x_2, y_2) , 则指针根据圆的参数方程的规律移动。圆的参数方程如下:

$$\begin{cases} x = a \sin(t) \\ y = a \cos(t) \end{cases}$$

其中 a 是圆的半径, t 参数(单位为度)。则 x_2 和 y_2 的值可用下式来计算:

$x := \text{Floor}(65 * \sin(t * 3.1415926 / 180));$ $x_2 := x_1 + x;$

$y := \text{Floor}((-1) * 65 * \cos(t * 3.1415926 / 180));$ $y_2 := y_1 + y;$

其中 t 为指针的当前度数。

【参考代码】

```
var
  Form1: TForm1;
  Num: integer;
  x, y: integer;
implementation
{$R *.dfm}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Form1.Canvas.Pen.Width := 3;
  if num=0 then                                     //第一次执行该事件
  begin
    x:=0;
    y:=-65;
    Form1.Canvas.Pen.Mode:=pmCopy;
    Form1.Canvas.Ellipse(5,5,155,155);             //画圆
    Form1.Canvas.MoveTo(80,80);
    Form1.Canvas.pen.Color :=Clred;
    Form1.Canvas.LineTo(80+x,80+y);                //画起始指针
  end
  else
  begin
    Form1.Canvas.pen.Color :=Clred;
    Form1.Canvas.Pen.Mode :=pmNotXor;
```

```

num:=num mod 360;
Form1.Canvas.MoveTo(80,80);           //擦除原来的指针
Form1.Canvas.LineTo(80+x,80+y);
x:=Floor(65*sin(Num*3.1415926/180));   //计算圆的参数
y:=Floor((-1)*65*cos(Num*3.1415926/180));
Form1.Canvas.Pen.Mode:=pmCopy;
Form1.Canvas.MoveTo(80,80);
Form1.Canvas.LineTo(80+x,80+y);       //重画指针
end;
num:=num+12;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    Num:=0;
end;

```

注意：由于使用了 Floor 函数，应当把“Math”单元包含进来。

12.3.2 上机练习二

【练习题目】：简单画图程序

编写一个能够根据用户的选择画椭圆和矩形的程序。程序的设计界面如图 12-15 所示，程序的运行界面如图 12-16 所示。程序运行时，默认画椭圆，也可以选择要画的图形。当在画图区域按下并拖动鼠标时，将出现一个相应图形的轮廓，显示在释放鼠标时所画图形的形状与大小，当松开鼠标的时候将创建一个内部填充色为蓝色的与图形轮廓一样大的相应图形。



图 12-15 程序设计界面

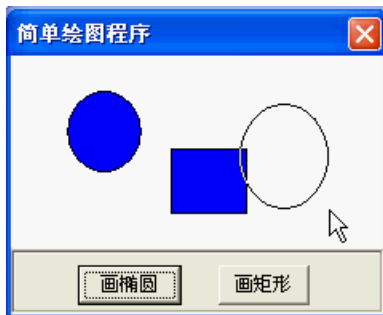


图 12-16 程序运行界面

【要点提示】

(1) 鼠标拖动时图形轮廓的显示。为显示图形轮廓，可设置两个 Tpoint 型的变量：StartPt 和 EndPt，分别用来存放鼠标按下时的坐标和当前坐标。可通过下述方法来实现“可擦写”的轮廓并画图：当按下鼠标时，程序将捕获所有的鼠标输入，并在名为 StartPt 的变量中记录鼠标的 (x,y) 坐标，同时用鼠标的 (x,y) 坐标初始化 EndPt 变量，然后设置画笔和笔刷的颜色，以画出所需图形的轮廓，把画笔的模式设置成 pmNot，这样画笔会以与底色相反的颜色绘图。现在，每当鼠标移动时，可两次画图：一次是在老地方擦写已画过的“可擦写”图形轮廓，

一次是在新位置上绘制出所需要的图形轮廓，然后在变量 EndPt 中记录鼠标新位置 (x,y) 坐标。最后当用户释放鼠标时，擦除上次的“可擦写”图形，并以最终的色彩画出图形来。

(2) 识别用户要画的图形。为了识别用户要画的图形，可定义一个整型变量 (Shape Kind)，在选择图形时 (发生按钮的 Click 事件)，给该整型变量赋不同的值：椭圆为 1、矩形为 2。在绘制图形时，根据该变量的值就知道应绘制什么样的图形。

【参考代码】

```
implementation
var
  StartPt, EndPt: TPoint;
  Capture: boolean;
  shapekind: integer;           //表示画图的种类，1 表示椭圆，2 表示矩形
{$R *.dfm}
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  StartPt.X:=X;                //设定起始点
  StartPt.Y:=Y;
  EndPt:=StartPt;
  Capture:=True;               //捕获鼠标操作
  With Image1.Canvas do
  begin
    pen.Mode:=pmNot;           //设定的画笔的模式为 pmNot，两次画同一点，等于没有画
    Pen.Color:=clBlack;        //画笔的颜色为黑色
    Brush.Style:=bsClear;       //画刷的填充风格为无
  end;
end;
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if (Capture) and ((EndPt.X<>X) or (EndPt.Y<>Y)) Then //如果鼠标发生了移动
  begin
    case shapekind of          //判断要画的是哪一种图形
    1: begin
      Image1.Canvas.Ellipse(StartPt.X, StartPt.Y, EndPt.X, EndPt.Y);
      //擦除刚才的椭圆轮廓
      Image1.Canvas.Ellipse(StartPt.X, StartPt.Y, X, Y); //重画新的椭圆轮廓
    end;
    2: begin
      Image1.Canvas.Rectangle(StartPt.X, StartPt.Y, EndPt.X, EndPt.Y);
      //擦除刚才的矩形轮廓
      Image1.Canvas.Rectangle(StartPt.X, StartPt.Y, X, Y); //重画新的矩形轮廓
    end;
  end;
  EndPt.X:=X;                 //设置新的结束点位置
```

```

        EndPt.Y:=y;
    end;
end;
procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);      //消除最后的轮廓线并画出需要的图形
begin
    Capture:=False;                          //取消对鼠标的捕获
    With Image1.Canvas do
    begin
        case Shapekind Of                  //判断要画的是哪一种图形, 并把鼠标轮廓线去除
            1: Ellipse(StartPt.X,StartPt.Y,EndPt.X,EndPt.Y);
            2: Rectangle(StartPt.X,StartPt.Y,EndPt.X,EndPt.Y);
        end;
        Pen.Mode:=pmCopy;                  //设置画笔的模式
        Brush.Color:=clBlue;                //设置画刷的填充色为蓝色
        Brush.Style:=bsSolid;
        case Shapekind Of                  //判断要画的图形种类, 绘制出需要的图形
            1: Ellipse(StartPt.X,StartPt.Y,X,Y);
            2: Rectangle(StartPt.X,StartPt.Y,X,Y);
        end;
    end;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    ShapeKind:=1;                          //开始认为要画的图形为椭圆
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Shapekind:=1;                          //画椭圆
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    Shapekind:=2;                          //画矩形
end;
end;

```

课后考场

一、选择题 (20 分, 每题 5 分)

1. 要想把窗体上的某一点设置为蓝色, 应使用 TCanvas 对象的_____属性。
A. Pen B. Brush C. Pixels D. CopyMode
2. 在绘制封闭图形时, 要想让它的填充色为某一指定的色彩, 应设置 TCanvas 对象的_____属性。
A. Pen.Color B. Brush.Color C. Pen.Style D. Brush.Style
3. 要想在窗体上绘制一段圆弧, 应使用 TCanvas 对象的_____方法。

A. Arc B. Draw C. Ellipse D. Pie

4. 要想使 TShape 组件显示圆角矩形, 应设置它的_____属性。

A. Pen B. Shape C. Brush D. Picture

二、填空题 (40 分, 每空 5 分)

1. 在 Form1 上画图时, 如果希望绘制出来的线条宽度为 5 个像素, 则应该首先执行的一条语句为_____。

2. 要把 Form1 的 Canvas 对象的第 5 行的第 6 个像素设置为红色, 执行的语句为_____。

3. 要在 Form1 窗体上绘制矩形, 应调用 TCanvas 对象的_____方法。

4. 要把名为 “D:\A1.BMP” 图片文件显示在 Image1 组件中, 应执行的语句是_____。

5. 要使 TImage 组件中显示的图片能够自动伸缩以填满图像框, 应把它的_____属性设置为 True。

6. _____对象可以包含一个位图图像, 拥有位图图像和调色板的句柄, 可以自动管理调色板。

7. 以下语句的作用是把 Form1 窗体的 Canvas 的画笔风格设置为点划线:

Form1.Canvas._____:= PsDashDot;

8. _____对象是 TBitmap、TIcon、TMetafile 对象的抽象基类, 如果不知道图像的具体类型, 可以使用它来存放。

三、程序设计题 (40 分, 每题 20 分)

1. 编写一个用户用鼠标画图的程序, 程序运行时, 用户按住鼠标并移动时, 将沿着鼠标的轨迹画线, 线的宽度为 5 个像素, 线的颜色为蓝色。程序的运行界面如图 12-17 所示。

2. 编写一个照片浏览程序。程序运行时, 将自动打开并显示当前目录下的名为 “Default.jpg” 的照片文件。用户可以通过单击窗体弹出一个【打开】对话框, 从中选择要显示的照片。照片刚好填满整个窗体, 当窗体改变大小时, 照片也改变大小以填满整个窗体。程序的运行界面如图 12-18 所示。



图 12-17 编程题一程序运行界面



图 12-18 编程题二程序运行界面

第 13 章 多媒体应用程序开发

本章要点

- 多媒体的基本概念
 - Delphi 7 开发多媒体应用程序的方法
 - TAnimate 组件的常用属性、方法及其使用
 - TMediaPlayer 组件的常用属性、方法、事件及其使用
 - 背景音乐的循环播放
-

13.1 理论知识

13.1.1 多媒体的概念

媒体是指信息的载体，主要分成两个方面：存储信息的媒体和表现信息的媒体。存储信息的媒体主要有软盘、硬盘、光盘等，而表现信息的媒体主要有文字、图形、图像、影片、动画、声音和音乐等。我们所讲的多媒体中的媒体主要是指表现信息的媒体。

所谓多媒体是指多种媒体的有机结合，是通过计算机的多媒体技术来实现的。“多媒体技术”在某种程度上已经成为声、文、图、动画、影像等媒体信息在计算机系统中综合应用的代名词。

多媒体技术具有下面的两个重要特点。

（1）多种媒体的综合性

单一的文本、声音或图形图像不能称为多媒体，只有把多种媒体有机地综合在一起才能称为多媒体。

（2）交互性。

多媒体的一个重要特点就是交互性，这也是它与广播电视及其他电教设备的本质不同之处。多媒体程序在运行过程中，能够根据用户的反应做出不同的响应。如考试多媒体软件，用户对某一道题做了答案后，如果正确，计算机则给出一种应答，如果错误，计算机也立即给出另一种应答。

计算机处理的媒体信息都是以文件的形式出现的。各类主要媒体文件及扩展名如下。

图像文件：位图文件（.BMP）、JPEG 文件（.JPG）。

声音文件：声波文件（.WAV）、MIDI 音乐文件（.MID）。

视频文件：AVI 文件（.AVI）、MPEG 文件（.MPG）。

在 Delphi 7 中，图像文件的显示是通过 TImage 等组件来实现的，另外在 Delphi 7 中还提

供了一个 TMediaPlayer 组件，利用该组件不但能够对波形声音、MID 音乐、视频等媒体文件进行播放和控制，而且能够控制多种多媒体设备。TMediaPlayer 组件位于 System 选项卡中，如图 13-1 所示。



图 13-1 TMediaPlayer 组件

另外 Delphi 7 还提供了一个控制多个图片连续播放的动画组件——TAnimate，该组件位于 Win32 选项卡中，如图 13-2 所示。



图 13-2 TAnimate 组件

13.1.2 TAnimate 组件的使用

1. TAnimate 组件的常用属性

- ┌ Active 属性：用于设置是否播放动画。值为 True 时，表示播放动画，值为 False 时，表示没有播放。
- ┌ AutoSize 属性：用于决定 TAnimate 组件的大小是否随着播放的动画图像的大小而改变，取值为 True 时，其大小随着动画图像的大小而自动改变，取值为 False 时，大小不改变。
- ┌ Center 属性：用于设置播放的动画图像是否在 TAnimate 组件的中央。值为 True 时，表示播放的动画图像在 TAnimate 组件的中央，值为 False 时，表示播放的动画图像在 TAnimate 组件的左上部分。
- ┌ FileName 属性：用于指明要播放的 AVI 的文件名（包括文件所在的路径）。如果设置了 FileName 属性的值，那么 CommonAVI 会自动指定为 aviNone。注意：该组件只能播放无声 AVI 文件。
- ┌ CommonAVI 属性：用于设定播放 Windows 自带的 AVI 动画，Windows 自带了一些和文件操作有关的 AVI 动画，如文件的删除、复制等。该属性的取值及含义如表 13-1 所示。

表 13-1 Commonavi 属性的取值及其含义

属 性 值	含 义
aviNone	不指定 AVI 文件，使用 Filename 中指定的 AVI 文件来播放
aviCopyFile	播放复制文件的 AVI 文件
aviCopyFiles	播放复制多个文件的 AVI 文件
aviDeleteFile	播放直接删除文件的 AVI 文件

续表

属 性 值	含 义
aviEmptyRecycle	播放清空回收站的 AVI 文件
aviFindComputer	播放搜索计算机的 AVI 文件
aviFindFile	播放搜索文件的 AVI 文件
aviFindFolder	播放搜索文件夹的 AVI 文件
aviRecycleFile	播放删除文件到回收站的 AVI 文件

- ┃ **StartFrame** 属性：用于设置从哪一帧开始播放，如果设置为 1，表示从第一帧开始，默认值为 1。
- ┃ **StopFrame** 属性：用于设置播放到动画的哪一帧结束，如果该属性值大于 AVI 文件中的总帧数，将播放到文件结束。
- ┃ **FrameCount** 属性：用于表示播放的 AVI 动画文件的总帧数，为只读属性。
- ┃ **Repetitions** 属性：用于设置重复播放 AVI 文件的次数。默认值为 0，表示无限制地播放。此时可通过把 **Active** 属性设为 **True** 来播放动画。
- ┃ **Transparent** 属性：用于设置播放背景是否为透明的。值为 **True**（默认）时，播放背景透明，值为 **False** 时，播放背景不透明。

2. TAnimate 组件的常用方法

（1）**Play** 方法：用于播放 AVI 文件。语法格式及功能如下。

[格式]：

```
procedure Play(FromFrame, ToFrame: Word; Count: integer);
```

[功能]：播放 AVI 文件中的若干帧。其中参数 **FromFrame** 表示播放的起始帧，**ToFrame** 表示播放的结束帧，**FromFrame** 的值应小于 **ToFrame** 的值。**Count** 表示播放次数。

（2）**Seek** 方法：用于定位到 AVI 动画文件的某一帧。语法格式及功能如下。

[格式]：

```
procedure Seek(Frame: SmallInt);
```

[功能]：定位到 AVI 动画文件的某一帧，参数 **Frame** 表示要定位到的帧数。

（3）**Reset** 方法：用于把 **StartFrame** 属性和 **StopFrame** 属性恢复成初始值。语法格式及功能如下。

[格式]：

```
procedure Reset;
```

[功能]：把 **StartFrame** 属性和 **StopFrame** 属性恢复成初始值。

（4）**Stop** 方法：用于终止 AVI 文件的播放。语法格式及功能如下。

[格式]：

```
procedure Stop;
```

[功能]：终止 AVI 文件的播放。

【例 13-1】 利用 **TAnimate** 组件播放 Windows 的预设动画。要求能够任意选择要播放

的预设动画种类，并能够设置是否循环播放。程序运行的界面如图 13-3 所示。

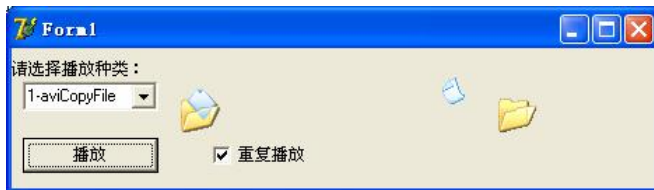


图 13-3 程序运行界面

【实现分析】

使用一个下拉式列表框列出 Windows 的所有预设动画，程序运行时可通过从下拉式列表框中选中一种预设动画，并通过复选框来设置是否循环播放，然后单击【播放】按钮即可播放。需要注意的是，从列表框中选中的值是字符串型的，必须转换成相应的常数才能赋值给 TAnimate 组件的 CommonAVI 属性。为便于转换，可在组合框的每个选项之前加上一个编号字符，在转换的时候通过函数分离出该编号字符，再根据它把相应的预设动画赋值给 TAnimate 组件的 CommonAVI 属性。播放动画时可通过把它的 Active 属性值设置为 True 来实现。

【界面设计】

本例组件对象及其属性设置如表 13-2 所示。

表 13-2 例 13-1 组件对象及其属性设置

对象名	属性名	属性值	说明
Label1	Caption	'请选择播放种类'	
	Font.Size	16	
	AutoSize	False	
ComboBox1	Items	'1-aviCopyFile' '2-aviCopyFiles' '3-aviDeleteFile' '4-aviEmptyRecycle' '5-aviFindComputer' '6-aviFindFile' '7-aviFindFolder' '8-aviNone' '9-aviRecycleFile'	设置组合框中显示的选项
	Style	'csDropDown'	组合框为下列式列表框
Animate1	CommonAVI	aviFindFolder	设定播放的默认动画
Button1	Caption	'播放'	单击它将播放选定的动画
CheckBox1	Caption	'重复播放'	用来设置是否循环播放

【程序代码】

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if checkBox1.Checked then
        Animate1.Repetitions :=0
```

```
else
    Animate1.Repetitions :=1;
case StrToInt(Copy(ComboBox1.Text,1,1)) of
    1: Animate1.CommonAvi:=aviCopyFile;
    2: Animate1.CommonAvi:=aviCopyFiles;
    3: Animate1.CommonAvi:=aviDeleteFile;
    4: Animate1.CommonAvi:=aviEmptyRecycle;
    5: Animate1.CommonAvi:=aviFindComputer;
    6: Animate1.CommonAvi:=aviFindFile;
    7: Animate1.CommonAvi:=aviFindFolder;
    8: Animate1.CommonAvi:=aviRecycleFile;
end;
Animate1.Active :=True;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    ComboBox1.Text :='1-aviCopyFile';
end;
```

13.1.3 TMediaPlayer 组件的使用

1. TMediaPlayer 组件概述

通过 TMediaPlayer 组件可以对多媒体文件和多媒体设备进行控制，TMediaPlayer 组件由一系列的按钮组成，如图 13-4 所示。

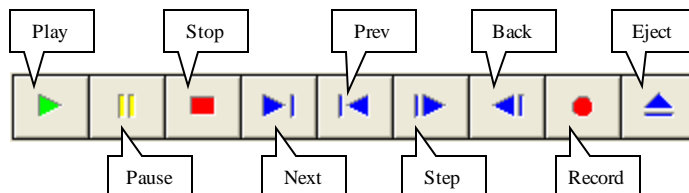


图 13-4 TMediaPlayer 组件及其按钮组成

TMediaPlayer 组件各按钮的功能如表 13-3 所示。

表 13-3 TMediaplayer 组件按钮及其功能

按钮名称	值	功能
Play	Btplay	播放媒体文件
Pause	Btpause	暂停或恢复播放或录制
Stop	Btstop	停止媒体文件的播放
Next	Btnext	跳到下一个磁道或者文件
Prev	Btprev	跳到前一个磁道或者文件
Step	Btstep	前进一个小段

续表

按 钮 名 称	值	功 能
Back	Btback	后退一个小段
Record	Btrecord	开始录制
Eject	Bteject	弹出（释放媒体文件）

2. TMediaPlayer 组件的常用属性

┌ AutoEnable 属性：该属性是一个 Boolean 型属性，其值为 True（默认值）时，媒体播放器可以根据 Mode 属性指定的当前模式和 DeviceType 属性指定的当前设备类型来确定哪些按钮可以使用。

注意：该属性值的设置优先于 EnableButton 属性的设置。

┌ AutoOpen 属性：用于确定当应用程序运行时，媒体播放器是否自动打开媒体设备。值为 True（默认值）时，在运行时可自动打开由 DeviceType 属性指定的多媒体设备（或若 DeviceType 为 dtAutoSelect 则打开由 FileName 属性指定的文件）；值为 False 时，则必须调用 Open 方法来打开设备。在打开设备时若出现错误，则出现类为 EMCIDeviceError 的意外处理。

┌ AutoRewind 属性：该属性为 Boolean 型属性，用于设置在多媒体文件播放完毕以后，是否自动返回到起点。值为 True（默认值）时，多媒体文件播放结束时能够自动返回到起点并重新开始播放。

┌ DeviceType 属性：用于设置要打开的多媒体设备的类型。它的默认值为 dtAutoSelect，即在程序运行时根据打开的多媒体文件（由 FileName 属性决定）的扩展名来确定文件的类型；若扩展名没有与之相关联的设备，则必须设置该属性来指定正确的设备类型。该属性的取值及含义如表 13-4 所示。

表 13-4 DeviceType 属性的取值及其含义

属 性 值	属 性 含 义
dtAutoSelect	根据要播放的多媒体文件名自动检测设备的类型
dtAVIVideo	打开 AVI 视频文件，该文件包括音频和视频，驱动程序为 AVI Video play
dtCDAudio	CD 唱盘，驱动程序为 CD Audio Player，一般从 CD-ROM 驱动器中播放
dtDAT	数字音频磁带，驱动程序为 Digital Audio Tape Player
dtDigitalVideo	数字视频（AVI、MPG、MOV 文件），驱动程序为 Digital Audio Player for Windows
dtMMMovie	多媒体电影格式，驱动程序为 MM Move Player
dtOther	其他未确定的多媒体设备
dtOverlay	模拟视频
dtScanner	与 PC 机相连的扫描仪
dtSequencer	乐器数据化设备（播放 MIDI 文件），驱动程序为 MIDI Sequencer for Windows
dtVCR	与 PC 机相连的录像机
dtVideodisc	与 PC 机相连的视频播放机
dtWaveAudio	波形音频，用来打开和播放 WAV 文件

┌ Display 属性：用于指定一个窗口作为多媒体文件的输出屏幕。该属性的默认值为 Nil，此时媒体播放器自动打开一个窗口用来显示输出。也可以把其他组件名（如 Panel 组件）

赋值给该属性，从而实现在其他组件的窗口中播放视频。

- **EnabledButtons** 属性：用于设置多媒体播放器组件中哪些按钮是有效的，哪些是无效的。它可以对九个按钮中的任何一个按钮进行设置。
- **FileName** 属性：用于为多媒体播放设备指定一个待播放的多媒体文件名。如果是录制文件，指定的是保存文件的文件名。主要用来设置三类文件：音波文件（.WAV）、数字化音乐文件（.MID）和视频文件（.AVI）。
- **Visible** 属性：用于决定 TMediaPlayer 组件是否可见，默认值是 True，表示是可见的。
- **VisibleButtons** 属性：用于决定 TMediaPlayer 组件中的各个按钮是否可见，可设置每个按钮的可见状态。
- **Mode** 属性：用于返回多媒体设备的当前状态。其取值及其含义如表 13-5 所示。

表 13-5 Mode 属性值及含义

属 性 值	属 性 含 义
mpNotReady	多媒体设备未准备好
mpStopped	多媒体设备处于停止状态
mpPlaying	多媒体设备处于播放状态
mpRecording	多媒体设备处于录制状态
mpSeeking	多媒体设备处于搜索状态
mpPaused	多媒体设备处于暂停状态
mpOpen	多媒体设备处于打开状态

- **Notify** 属性：用于决定当一个多媒体控制方法（比如 Open、Play 等）完成以后，是否响应下一个多媒体方法。若它的值为 True，表示接受下一个多媒体控制方法，并触发 onNotify 事件；若它的值为 False，表示不接受下一个多媒体控制方法，也不会触发 onNotify 事件。
- **NotifyValue** 属性：用于表示上一个控制命令的执行情况。它的取值详见表 13-6。

表 13-6 NotifyValue 属性的值及含义

属 性 值	属 性 含 义
Nvsuccessful	控制命令成功
NvSuperseded	控制命令被其他的命令所代替
Nvaborted	控制命令被用户所终止
NvFailure	控制命令失败

- **TimeFormat** 属性：用于设置描述时间的格式，该属性是一个运行属性。它的取值及其含义如表 13-7 所示。其中两个比较重要的值是：“tfMillisecond”表示多媒体时间计量单位为毫秒；“tfFrames”表示多媒体时间计量单位为帧。

表 13-7 TimeFormat 属性的值及含义

属 性 值	属 性 含 义
tfMilliseconds	毫秒
tfHMS	小时、分钟和秒
tfMSF	分种、秒和帧数时间存储格式
tfFrames	帧数
tfSMPTE24	小时、分钟、秒、基于每秒 24 帧的帧数
tfSMPTE25	小时、分钟、秒、基于每秒 25 帧的帧数
tfSMPTE30	小时、分钟、秒、基于每秒 30 帧的帧数
tfSMPTE30Drop	小时、分钟、秒、基于每秒 30 帧的 Drop 帧数
tfBytes	字节数
tfSamples	取样数
tfTMSF	音轨、分钟、秒和帧数

- ┆ Tracks 属性：用于表示 CD 音乐总轨数。
- ┆ TrackLength 属性：用于表示音乐轨的长度。
- ┆ TrackPosition 属性：用于表示当前音乐轨的播放位置。
- ┆ Length 属性：用于表示播放媒体的总长度。长整型。
- ┆ Position 属性：用于表示媒体当前的播放位置。长整型。
- ┆ Wait 属性：该属性为 Boolean 类型，用于决定是否当媒体控制方法执行结束后，才将控制权交给应用程序。当这个属性设置为 True 时，必须等到媒体控制方法执行结束后，才将控制权交给应用程序；当这个属性设置为 False 时，程序不等前面一个媒体控制方法执行完就继续执行下面的程序代码。
- ┆ Frames 属性：用于确定调用 Back 或 Step 方法时，前进或后退的帧数。

3. TMediaPlayer 组件的常用方法

TMediaPlayer 组件的常用方法及其作用如表 13-8 所示。

表 13-8 TMediaPlayer 组件的常用方法及其作用

名 称	作 用
Next	跳到下一个轨道的开始，如果媒体不用轨道，将跳到媒体的结尾
Open	打开多媒体设备
Pause	暂停多媒体设备
Previous	跳到前一个轨道的开始，如果媒体不用轨道，将跳到媒体的开始
Stop	停止多媒体设备的播放
Play	播放多媒体文件或轨道
Back	在当前打开的媒体中，向后移动一定数量的帧，移动的帧数由 Frames 属性决定
Rewind	回到播放的起点
Close	关闭多媒体设备
StartRecording	开始录制
Eject	弹开多媒体设备，并释放媒体

续表

名 称	作 用
Step	在当前打开的媒体中，向前移动一定数量的帧，移动的帧数由 Frames 属性决定
Resume	恢复暂停的多媒体播放或记录操作
Save	保存已经打开的媒体信息

4. TMediaPlayer 组件的常用事件

TMediaPlayer 组件能够响应 OnClick、OnPostClick、OnEnter、OnExit、OnNotify 等多种事件，下面重点介绍 OnNotify 事件的使用。

当 Notify 属性设置为 True 时，每当多媒体控制方法（如前面所讲的 Open, Save, Back, Close, Eject, Pause 等方法）完成以后，就会产生 OnNotify 事件。

注意：在响应了 OnNotify 事件以后，Notify 属性必须重新设置为 True，以便系统再次产生 OnNotify 事件。

【例 13-2】 编写一个波形音频文件的播放程序。程序的设计界面如图 13-5 所示。程序运行时，如图 13-6 所示。单击【打开】按钮将出现一个打开对话框，供用户选择一个波形音频文件。选择文件名后单击【播放】按钮将播放该波形音频文件。



图 13-5 例 13-2 程序设计界面



图 13-6 例 13-2 程序运行界面

【实现分析】

通过一个打开对话框组件来选择文件，选择文件的文件名保存在打开对话框组件的 FileName 属性中，然后可把这个 FileName 属性值赋值给 TMediaPlayer 组件的 FileName 属性，从而使 MediaPlayer 组件能够播放该文件。一般情况下给 TMediaPlayer 组件的 FileName 属性赋值前先关闭多媒体设备，赋值后再打开多媒体设备。播放可以通过调用 TMediaPlayer 组件的 Play 方法来实现。

【界面设计】

本例组件对象及其属性设置如表 13-9 所示。

表 13-9 例 13-2 组件属性设置及组件作用

对 象 名	属 性 名	属 性 值	作 用
MediaPlayer1	Visible DeviceType	False dtAutoSelect	播入选择的波形音频
OpenDialog1	Filter	'WAV 文件*.WAV'	选择要播放的波形音频文件
Button1	Caption	'打开'	单击它将出现打开对话框供用户选择一个波形音频文件
Button2	Caption	'播放'	单击它将播放选择的波形音频文件

【程序代码】

主要程序代码如下：

```
procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:string;           //存放应用程序所在目录的目录名
begin
    GetDir(0,CurDir);        //获取应用程序所在的当前目录的目录名
    MediaPlayer1.FileName :=CurDir+'W1.wav';    //形成要播放的默认路径名
    MediaPlayer1.Open ;      //打开多媒体设备
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog1.Execute ;           //显示打开对话框
    if Opendialog1.FileName="" then      //如果没有选择文件
        ShowMessage('你没有选择声音文件') //提示
    else
        begin
            MediaPlayer1.Close;          //先关闭多媒体设备
            MediaPlayer1.FileName:=OpenDialog1.FileName ; //设置要播放的多媒体文件名
            MediaPlayer1.Open;            //打开多媒体设备
        end;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    MediaPlayer1.Play ;                 //播放多媒体文件
end;
```

13.2 典型实例

13.2.1 典型实例一

【实例题目】：唐诗朗诵程序的设计

编写一个选择唐诗并朗读的多媒体程序。程序的设计界面如图 13-7 所示。程序运行时，当鼠标移动到相应的唐诗标题上时，标题将出现背景色，如图 13-8 所示。此时若单击该唐诗标题将朗诵该唐诗。单击**【退出】**按钮将退出该程序的运行。

【实现方法】

当单击相应标签时，播放相应的唐诗，唐诗的录音必须是已经存放好的 WAV 文件，本题的文件名分别为 1.WAV, 2.WAV, 3.WAV 和 4.WAV 等。当发生相应标签的 Click 事件时，应首先关闭多媒体设备，然后把相应的唐诗文件名赋值给 MediaPlayer1 的 Filename 属性，然后调用 MediaPlayer1 的 Open 方法打开多媒体文件，最后再调用 MediaPlayer1 的 Play 方法播放该文件。另外背景色是否显示可通过设置标签组件的 Transparent 属性来实现。



图 13-7 程序设计界面



图 13-8 程序运行界面

【界面设计】

本例组件属性设置及组件作用如表 13-10 所示。

表 13-10 窗体及组件的属性设置

对 象 名	属 性 名	设 置 值	对 象 作 用
Label1	Caption Color Font.Size Transparent AutoSize	'咏 鹅' clNavy 14 True False	显示唐诗名，单击它播放相应唐诗
Label2	Caption Color Font.Size Transparent AutoSize	'春 晓' clNavy 14 True False	显示唐诗名，单击它播放相应唐诗
Label3	Caption Color Font.Size Transparent AutoSize	'春夜喜雨' clNavy 14 True False	显示唐诗名，单击它播放相应唐诗
Label4	Caption Color Font.Size Transparent AutoSize	'登鹤鹊楼' clNavy 14 True False	显示唐诗名，单击它播放相应唐诗
MediaPlayer1	Visible	False	播放唐诗
Button1	Caption	'退出'	单击它退出应用程序

【程序代码】

```

var
    Form1: TForm1;
    cdir:string;           //存放当前目录
implementation
{ $R *.dfm}
procedure TForm1.Label1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);

```

```
begin
    Label1.Transparent :=False;           //设置 Label1 标签不透明
end;
procedure TForm1.Label2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Label2.Transparent :=False;           //设置 Label2 标签不透明
end;
procedure TForm1.Label3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Label3.Transparent :=False;           //设置 Label3 标签不透明
end;
procedure TForm1.Label4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Label4.Transparent :=False;           //设置 Label4 标签不透明
end;
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Label1.Transparent :=True;             //把所有标签组件的属性均设为透明
    Label2.Transparent :=True;
    Label3.Transparent :=True;
    Label4.Transparent :=True;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
    getdir(0,Cdir);                         //得到当前目录
end;
procedure TForm1.Label1Click(Sender: TObject);
begin
    MediaPlayer1.Close ;                   //关闭多媒体设备
    mediaplayer1.FileName:=cdir+'\1.wav';  //形成要播放的 WAV 文件名
    mediaplayer1.open;                     //打开多媒体设备
    mediaplayer1.play;                      //播放唐诗
end;
.....
//以下省略了 Label2Click、Label3Click、Label4Click 事件代码，它们的作用与 Label1Click 类似
procedure TForm1.Button1Click(Sender: TObject);
begin
    mediaplayer1.Close ;                   //关闭多媒体设备
    application.Terminate ;                //关闭应用程序
end;
```

13.2.2 典型实例二

【实例题目】：视频文件播放控制器

编写一个视频文件播放控制程序，程序的设计界面如图 13-9 所示，程序的运行界面如图 13-10 所示。程序运行时，单击【打开】按钮将出现一个打开对话框让用户选择一个视频文件，单击相应的按钮将实现相应的操作。有两点需要注意：一是在单击【暂停】按钮暂停播放时，【暂停】按钮的标题变为“恢复”，再单击该按钮将恢复视频文件的播放；二是单击【前进】或【后退】按钮时，视频文件将前进或后退 5 帧。



图 13-9 程序设计界面



图 13-10 程序运行界面

【实现方法】

可使用一个 TOpenDialog 对话框组件来选择文件，使用 Open 方法来打开多媒体设备，使用 Play 方法播放多媒体文件，使用 Stop 方法来停止多媒体设备的播放。为实现“前进”和“后退”5 帧的功能，可把 TMediaPlayer 组件的 Frames 属性设置为 5，然后调用 Step 或 Back 方法。

【界面设计】

本例组件属性设置及组件作用如表 13-11 所示。

表 13-11 组件属性设置及组件作用

对 象 名	属 性 名	设 置 值	对 象 作 用
MediaPlayer1	Visible	False	播放视频文件
Panel1	Caption	"	作为视频文件播放的窗口
OpenDialog1	Filter	*.AVI *.AVI	用来选择视频文件
Button1	Caption	'打开'	选择视频文件并打开多媒体设备
Button2	Caption	'播放'	播放视频文件
Button3	Caption	'前进'	前进 5 帧
Button4	Caption	'后退'	后退 5 帧
Button5	Caption	'暂停'	暂停或恢复视频文件的播放
Button6	Caption	'停止'	停止视频文件的播放
Button7	Caption	'结束'	结束当前应用程序

【程序代码】

```
procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:string;
begin
    GetDir(0,CurDir);
    MediaPlayer1.FileName :=CurDir+'\A2.AVI';           //设置默认播放的视频文件名
    MediaPlayer1.Display :=Panel1;
    MediaPlayer1.Open;
    OpenFileDialog1.InitialDir :=CurDir;               //设置打开对话框的初始目录
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog1.Execute;                           //调用打开对话框
    MediaPlayer1.Stop;                                  //停止播放
    MediaPlayer1.FileName:=OpenDialog1.FileName;        //设置播放的文件名
    MediaPlayer1.Frames :=5;                            //设置前进或后退的帧数
    MediaPlayer1.Display:=Panel1;                       //设置视频文件的播放窗口
    MediaPlayer1.Open;                                   //打开多媒体设备
end;
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    MediaPlayer1.Stop;                                  //停止多媒体设备的播放
    MediaPlayer1.Close;                                  //关闭多媒体设备
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    MediaPlayer1.Play;                                  //播放
end;
procedure TForm1.Button5Click(Sender: TObject);
begin
    if Button5.Caption = '暂停' then
    begin
        MediaPlayer1.Pause;                             //暂停播放
        Button5.Caption :='恢复';
    end
    else
    begin
        MediaPlayer1.Resume ;                            //恢复播放
        Button5.Caption :='暂停';
    end
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    MediaPlayer1.Step ;                                  //前进若干帧
```

```
end;  
procedure TForm1.Button4Click(Sender: TObject);  
begin  
    MediaPlayer1.Back;                               //后退若干帧  
end;  
procedure TForm1.Button6Click(Sender: TObject);  
begin  
    MediaPlayer1.Stop;                               //停止播放  
end;
```

13.3 上机练习

13.3.1 上机练习一

【练习题目】：为唐诗朗诵程序添加背景音乐

在典型实例一的基础上增加播放背景音乐的功能：要求背景音乐可以选择，并且可以循环播放。程序的设计界面如图 13-11 所示，程序的运行界面如图 13-12 所示。程序运行时单击**【选择背景音乐】**按钮，将出现一个**【打开】**文件对话框让用户选择背景音乐文件，选择过后将自动进行背景音乐的播放，并且背景音乐能够循环播放。

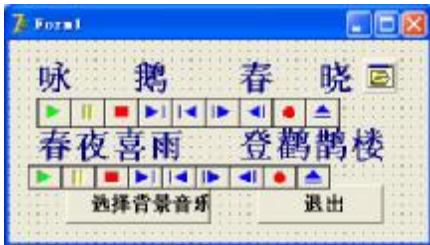


图 13-11 程序设计界面

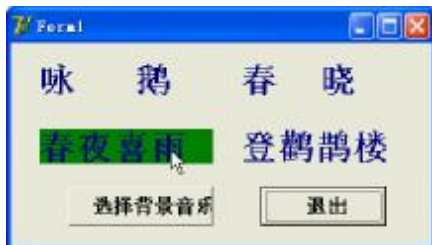


图 13-12 程序运行界面

【要点提示】

该上机练习有两个要点。

(1) WAV 文件和 MID 文件的同时播放。由于 WAV 和 MID 文件使用的是不同的多媒体设备，因此它们能够同时播放。但一般情况下不能同时播放两个 WAV 文件或两个 MID 文件。

(2) 背景音乐的循环播放。TMediaPlayer 组件有一个重要事件 OnNotify，该事件在命令执行成功、失败或中止时均会发生，并可以通过 TMediaPlayer 组件的 NotifyValue 属性来判断用户命令执行的情况。为使 MID 文件能够循环播放，可先设置 TMediaPlayer 的 AutoRewind 属性和 Notify 属性的值均为 True，然后在 OnNotify 事件中，判断命令是否正常结束，如果正常结束且媒体已经播放到文件尾，则从头开始播放。

【参考代码】

在典型实例一的基础上增加一个 MediaPlayer2 组件，用来循环播放背景音乐。增加的参

考程序代码如下:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  getdir(0,Cdir);
  MediaPlayer2.Notify :=true ;
  MediaPlayer2.FileName:=cdir+'\1.mid';           //形成默认的背景文件
  MediaPlayer2.Open ;                             //打开播放背景音乐的多媒体设备
  MediaPlayer2.Play ;                             //播放背景音乐
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  Opendialog1.Execute ;                           //弹出打开对话框
  MediaPlayer2.Close ;                            //关闭背景音乐的播放
  if (length(Opendialog1.FileName )<>0) then      //如果选了文件名
    MediaPlayer2.FileName:=Opendialog1.FILENAME; //设置播放的背景音乐文件名
  MediaPlayer2.Open;                              //打开播放背景音乐的设备
  MediaPlayer2.Play ;                             //播放背景音乐
end;
procedure TForm1.MediaPlayer2Notify(Sender: TObject);
begin
  if MediaPlayer2.Notify Value =nvSuccessful then //如果命令执行成功
  begin
    MediaPlayer2.Position :=0;                    //从头开始进行播放
    MediaPlayer2.Notify :=True;
    MediaPlayer2.Play ;                           //播放
  end;
end;
```

13.3.2 上机练习二

【练习题目】: 显示视频文件的播放进度

对典型实例二进行改造, 使它能够显示出视频文件的播放进度。程序的设计界面如图 13-13 所示, 程序的运行界面如图 13-14 所示。程序运行时将在一个进度条中显示视频文件播放的进度。

【要点提示】

为了能够显示多媒体文件的播放进度, 可以使用一个 `TProgressBar` 组件和一个 `TTimer` 组件。在多媒体文件播放前把 `TProgressBar` 的 `Min` 属性值设置为 0、`Max` 属性值设置为多媒体文件的长度 (`TMediaPlayer` 组件的 `Length` 属性值), 然后在时钟组件的 `Timer` 事件中用 `TMediaPlayer` 组件的当前播放位置 (`Position` 属性值) 去更新 `TProgressBar` 组件的 `Position` 属性值即可。



图 13-13 程序设计界面



图 13-14 程序运行界面

【参考代码】

```

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    //用 MediaPlayer1.Position 属性更新 ProgressBar1.Position 的属性值
    ProgressBar1.Position :=MediaPlayer1.Position ;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog1.Execute;
    MediaPlayer1.Stop;
    MediaPlayer1.FileName:=OpenDialog1.FileName;
    MediaPlayer1.Frames :=5;
    MediaPlayer1.Display:=Panel1;
    MediaPlayer1.Open;
    ProgressBar1.Min :=0;           //设置进度条的 Min 属性值为 0
    ProgressBar1.Max :=MediaPlayer1.Length ;
                                   //设置进度条的 Max 属性值为多媒体文件的长度
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    MediaPlayer1.Step ;
    //用 MediaPlayer1.Position 属性更新 ProgressBar1.Position 的属性值
    ProgressBar1.Position :=MediaPlayer1.Position ;
end;
procedure TForm1.Button4Click(Sender: TObject);
begin
    MediaPlayer1.Back ;
    //用 MediaPlayer1.Position 属性更新 ProgressBar1.Position 的属性值
    ProgressBar1.Position :=MediaPlayer1.Position ;

```

```
end;
```

课后考场

一、选择题（20 分，每题 5 分）

1. 要使 TAnimate 组件播放动画，只需把它的_____属性设置为 True 即可。
A. Active B. AutoSize C. Center D. Play
2. 通过 TMediaPlayer 组件的_____属性可以设置要打开的多媒体设备的类型。
A. DeviceType B. AutoRewind C. Mode D. FileName
3. 通过 TMediaPlayer 组件的_____属性可以获知上一个控制命令的执行情况。
A. Mode B. Notify C. NotifyValue D. Wait
4. 要使 TMediaPlayer 组件暂停播放或录制，应调用它的_____方法。
A. Pause B. Stop C. Eject D. Step

二、填空题（40 分，每空 5 分）

1. 多媒体技术的两个最重要特点是_____和_____。
2. 要使 TAnimate 组件播放某个 AVI 文件，应把它的_____属性设置为 AVI 文件名。
3. 要使 TAniamte 组件从动画的第 5 帧开始播放，需把它的_____属性设置为 5。
4. 要使 TMediaPlayer 组件在播放视频时播放的窗口是 Panel1，应把它的_____属性设置为 Panel1。
5. 在用 TMediaPlayer 组件播放视频时，可以使用它的_____属性获取当前播放的位置。使用它的_____属性可以获取媒体文件的总帧数。
6. 当把 Notify 属性值设置为 True 时，每当多媒体控制方法完成以后，就会产生_____事件。

三、程序设计题（40 分，每题 20 分）

1. 编写一个能够播放视频和背景音乐的程序。视频和背景音乐可以由用户自己来选择。
2. 编写一个 CD 播放器程序。播放器的设计界面如图 13-15 所示。

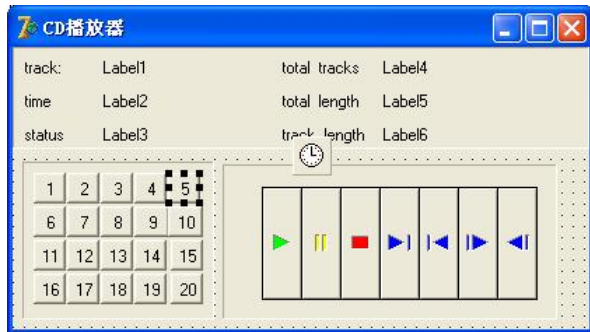


图 13-15 CD 播放器的设计界面

第 14 章 数据库应用开发

本章要点

- ▮ 数据库的基本概念
 - ▮ Delphi 中的 Database Desktop 应用程序的使用
 - ▮ TTable 组件的常用属性、方法、事件及其使用
 - ▮ TQuery 组件的常用属性、方法、事件及其使用
 - ▮ 常用的数据控制组件
 - ▮ SQL 语言编程
-

14.1 理论知识

14.1.1 数据库的基本概念

目前，人类社会已经进入了信息社会，用“信息爆炸”来描述这个社会的信息之多并不过分。对于非专业人员来说，没有必要区分信息和数据的概念，通常所说的信息处理也可以看成是数据处理，用计算机进行数据处理已经成为很多行业日常工作不可缺少的一个环节。数据库技术可以简单地理解为最新的数据处理技术，它已经深入到我们生活的方方面面。

1. 数据库

所谓数据库（Database，DB）其实就是存放在计算机的外存储器中的相关数据的集合，它是通过文件或类似于文件的数据单位组织起来的。数据库只是数据的集合，建立数据库的目的是为了使用数据库，为了对数据库中的数据进行存取，必须通过数据库管理系统（Database System Management，DBMS）。数据库管理系统是对数据进行管理的软件，是数据库系统的核心，数据库的一切操作，包括数据库的建立、数据的检索、修改、删除等操作，都是通过数据库管理系统来实现的。数据库管理系统只提供对数据的管理功能，为了实现某种具体的功能，必须要有相应的数据库应用程序。正是由于数据库应用程序的不同，才使数据库应用丰富多彩。例如，某单位的人事管理系统和财务管理系统均是由同一种数据库管理系统来完成的，但它们实现的功能是不一样的，原因就是它们的数据库应用程序不一样。

日常生活中见到的能够使用的与数据库有关的计算机系统均可认为是数据库系统。所谓数据库系统是指实际可运行的，按照数据库方式存储、维护和向应用系统提供数据或信息支持的计算机系统。它是在计算机系统中引进数据库后的系统构成，一个完整的数据库系统还应包括数据库管理员（Database Administrator，DBA），所谓数据库管理员是指一组熟悉计算机数据处理业务、负责设计和维护数据库的技术人员。

因此一个完整的数据库系统由数据库、数据库管理系统、数据库应用程序、计算机软件 and 硬件系统及 DBA 组成。

2. 数据模型与关系数据库

数据库中的数据是按照一定的数据模型组织的，数据模型是把现实世界转换为计算机能够处理的数据世界的桥梁。目前常用的数据模型有三种，分别是层次模型、网状模型和关系模型。由于层次模型和网状模型需要较深的数学基础，所以使用它们的人并不多，最常用的数据模型是关系模型。

在关系模型中，数据被组织成若干二维表的结构，每一个二维表称为一个关系或表。表中的一行称为一个元组，在计算机中存放则称为记录。表中的一列称为属性，在计算机中存放就是字段。关系的构成如图 14-1 所示。

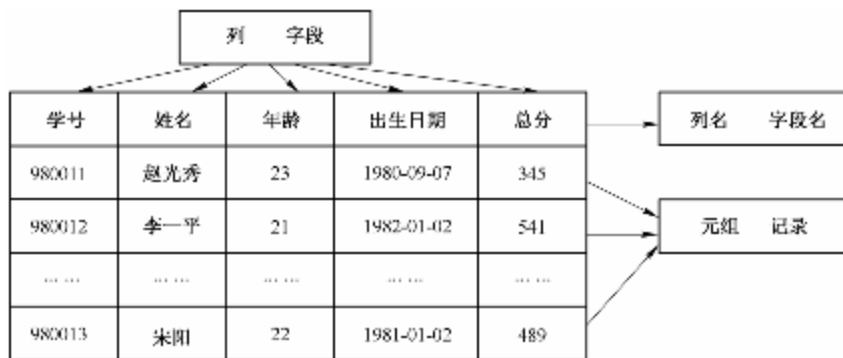


图 14-1 关系的构成

通常所说的关系数据库由相关的多个表组成。如要管理学生成绩，可做一个数据库，该数据库可由两个表组成：一个表用来存放学生的基本数据，如学号、姓名、年龄、年级、专业等信息；另一个表用来存放学生的选课信息，如学生学号、所选课的课号、选课的成绩等信息。

通过对表的分析，可得到下列有关表的性质：

- (1) 表中的每一列均不可再分；
- (2) 表中的每一列数据的数据类型是相同的；
- (3) 表中的两列不能取相同的名字；
- (4) 表中不允许有完全相同的两行，即任两条记录必须能够区分；
- (5) 交换行和列的顺序，不改变表的含义。

在关系数据库管理系统中，要建立一个数据库通常要经过以下几个步骤：

- (1) 建立数据库文件，数据库文件通常用来容纳各个表；
- (2) 建立表的结构，即建立表的各个字段的字段名、字段类型、字段宽度等；
- (3) 向表中录入实际的数据。

14.1.2 利用数据库桌面创建数据库

1. 数据库桌面的作用

数据库桌面 (Database Desktop) 是 Delphi 7 自带的一个数据库管理工具，它的主要作用

如下。

(1) 创建和维护数据库

数据库桌面应用程序可以直接创建和维护 Paradox 数据库,但不仅是用于创建 Paradox 表,几乎当前所有格式的数据库表均可以通过它来创建。但除了 Paradox 表以外,要创建其他格式的数据库表,计算机必须安装相应的数据库管理系统或驱动程序,否则在创建数据库表时将会出错。

使用数据库桌面时要特别注意区分数据库和数据库表的概念,数据库是数据库表的容器,对于 Access 数据库来说它是一个数据库文件,对于 Paradox 来说,它是一个容纳数据表的目录(文件夹)。数据库表是实际容纳数据的表格。

(2) 维护数据库别名

可以把数据库别名理解为存放数据库表的容器,对于像 Paradox 这类的数据库,一个文件就是一个数据表,因此数据库别名就是指一个存放数据库表的路径。假设有若干个 Paradox 表存放在 D:\P1 子目录下,就可以建立一个数据库别名(假设为 PP),该别名与 D:\P1 对应起来。一旦建立好了这种别名和路径或数据库名的对应关系,以后需要访问数据库表的时候,就不必再指定具体的路径或数据库名,只需指明别名就行了。

在数据库桌面应用程序中,可以实现别名的创建和删除等操作。

(3) 创建 SQL 文件和执行 SQL 命令

SQL 是结构化查询语言的英文缩写,利用它可以实现数据库的建立、维护和查询。利用数据库桌面应用程序可以创建和执行 SQL 文件。

2. 数据库的建立

这里通过建立一个实际的通讯录数据库来介绍在 Delphi 中建立数据库的方法。根据对目前通讯录信息的分析,可设计将建立的通讯录表的结构,如表 14-1 所示。

表 14-1 通讯录表的结构

字 段 名	字 段 类 型	字 段 宽 度	说 明
好友号	Alpha	8	Alpha 相当于字符型
姓名	Alpha	10	
好友类别	Alpha	10	
年龄	Short	系统默认为 2	短整型
出生日期	Date	由系统指定	日期型
通讯地址	Alpha	30	
邮政编码	Alpha	6	
电话号码	Alpha	15	
电子邮件	Alpha	15	
QQ 号码	Alpha	10	
交往大事	Memo	系统默认	备注型存放大量文字

Delphi 的本地数据库为 dBase 数据库和 Paradox 数据库,利用 Database Desktop 可以很方便地创建这两类数据库表。创建一个 Paradox 数据库表的步骤如下:

(1) 在“Database Desktop”应用程序中,执行【File】→【New】→【Table】命令,将

会出现如图 14-2 所示的【Create Table】(创建表)对话框。在【Table type】列表框中可以选择表的类型。

(2) 选择默认的类型“Paradox 7”，单击【OK】按钮，在弹出的对话框中可以定义表的结构，即表的每个字段的字段名、字段类型、字段宽度和主键等，并可建立索引，如图 14-3 所示。

(3) 在【Field Name】列下面输入字段名，如“好友号”，然后在【Type】列下面单击鼠标右键，将会出现如图 14-4 所示的【字段类型】下拉菜单，在该菜单中选中类型“Alpha”，然后在【Size】列下面单击，输入字段宽度为 8。关于字段类型的具体含义请参考有关 Paradox 数据库的书籍。如果要把“学号”字段设为主键，可在该行的【Key】列双击鼠标右键，这时将会出现一个“*”号，代表该列是主键。如果要取消主键，可再双击该处，“*”号将消失。



图 14-2 创建表对话框

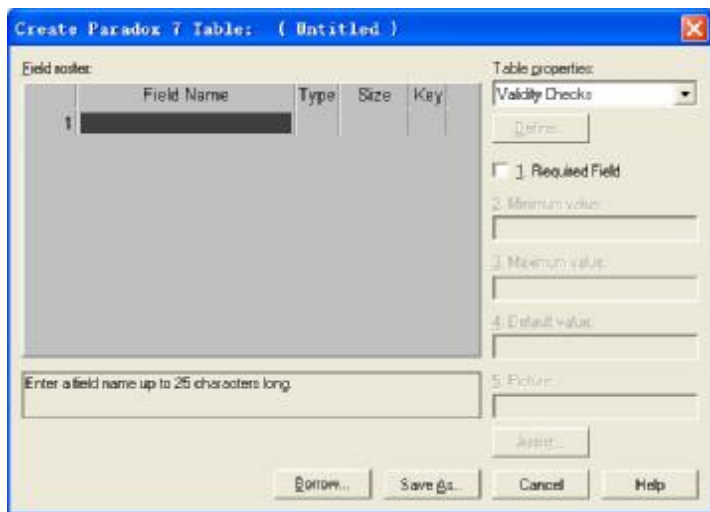


图 14-3 创建表结构对话框

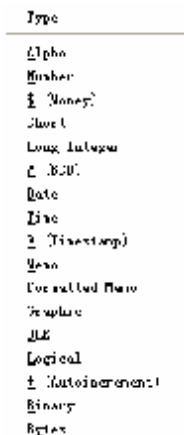



图 14-4 【字段类型】下拉菜单

常用的数据类型及其含义如下。

- ! Alpha: 字符串型，宽度为 1~255，由用户自己定义。
- ! Number: 数值型，宽度由系统自动指定。
- ! Date: 日期型，宽度由系统自动指定。
- ! Logical: 逻辑型，宽度由系统自动指定。
- ! Memo: 备注型，一般用来存放长度超过 255 个字符的数据，宽度设置范围也是 0~255，但是超过宽度范围部分的数据会自动存放到另外一个文件 (*.MB) 里。
- ! Graphic: 图片型，用来存放二进制位图信息，宽度由系统自动指定。

(4) 按 Enter 键把光标移到下一行，可接着创建下一个字段。依次创建多个字段，如图 14-5 所示。

(5) 还可以为表创建“辅助索引”(Secondary Indexes)，方法是在图 14-3 所示的创建表结构对话框中，在【Table Properties】列表框中选中“Secondary Indexes”，然后单击【Define】

按钮, 将会出现【Define Secondary Index】(定义辅助索引)对话框, 如图 14-6 所示。在该对话框中选中需要定义的索引字段, 如“姓名”, 然后单击  按钮, 把选中的字段移动到【Indexed fields】窗口中即可。辅助索引有四个选项, 其含义如下。

Field roster:				
	Field Name	Type	Size	Key
1	好友号	A	8	
2	姓名	A	10	
3	好友类别	A	10	
4	年龄	S		
5	出生日期	D		
6	通信地址	A	30	
7	邮政编码	A	6	
8	电话号码	A	15	
9	电子邮件	A	15	
10	QQ号码	A	10	
11	交往大事	M	100	

图 14-5 创建了多个字段

- ! **Unique:** 惟一索引。若选中该复选框, 则要求所有记录在索引字段上的取值都是不同的。在创建索引时, 或创建索引后再修改或输入数据时, 如果有两个记录在该字段上的值出现相同, 则将弹出一个错误提示信息。
 - ! **Descending:** 用来确定辅助索引是升序还是降序。不选中它表示升序, 选中它表示降序。
 - ! **Case sensitive:** 大小写字母敏感。选中它, 大小写字母认为是不一样的字母, 如果没有选中, 大小写字母不区分。
 - ! **Maintained:** 确定是否自动维护辅助索引。选中该复选框后, 每当表有所变化, 索引将自动更新。自动维护索引只能建立在定义了主键的表中。非自动维护索引当在引用它时才被更新。
- (6) 辅助索引属性设置好后, 单击【OK】按钮, 将会出现如图 14-7 所示的【Save Index As】(保存索引)对话框, 给索引取一个名字“XM”, 然后单击【OK】按钮, 索引定义完成。

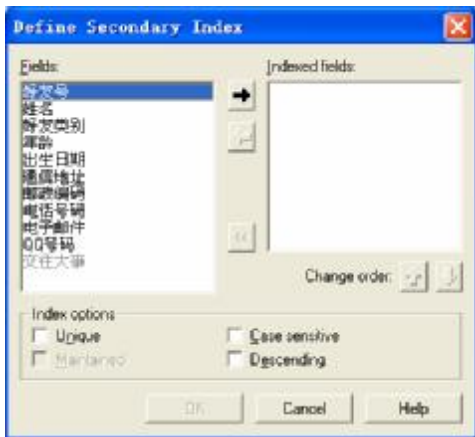


图 14-6 定义辅助索引对话框

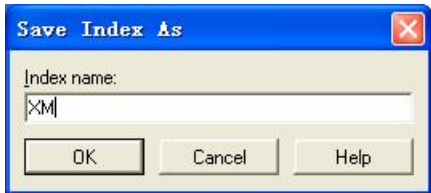


图 14-7 保存索引对话框

(7) 表结构创建好后, 单击【Save as】按钮, 将会出现如图 14-8 所示的【Save Table As】(保存表) 对话框。可以通过选中数据库别名和选择保存文件夹两种方式确定数据库表保存的位置。这里选择保存位置为 “D:\DelphiApp\14”, 表名为 “通讯录”, 设置完成后单击【保存】按钮, 表结构将保存起来。表结构创建完毕。



图 14-8 保存表对话框

(8) 数据库表的结构创建后, 应向其中输入数据。首先应打开数据库表, 方法是执行【File】→【Open】→【Table】命令, 在出现的【Open Table】(打开表) 对话框中找到相应的数据表 “通讯录”, 单击【打开】按钮, 然后按下工具栏上的 EditData 按钮 (即该按钮为按下状态), 就可以输入数据了。图 14-9 是输入了几行数据的通讯录表。当不希望输入数据时, 可再单击一次 EditData 按钮, 使该按钮处于弹起状态。从而结束数据的录入。

通讯录	好友号	姓名	好友类别	年龄	出生日期	通信地址	邮政编码	电话号码	电子邮件
1	00000001	胡光兵	同学	35	1969-10-3	南京市标营2号	210007	025-84536524	HgB@163.com
2	00000002	李建东	同学	35	1969-9-7	南京市标营2号	210007	025-43873434	Ljd@163.com
3	00000003	董晓宝	亲戚	20	1985-4-8	安徽省无为县宝山乡	320008	0565-8765645	Txb@sohu.com
4	00000004	龙在天	同事	25	1979-10-23	南京市标营2号	210007	025-38734743	lzt@263.com

图 14-9 输入了几条记录的通讯录表

3. 数据库别名的建立

数据库别名有两种: 公共别名 (Public Alias) 和项目别名 (Project Alias)。本书只介绍公共别名, 该别名保存在 BDE 配置文件 IDAPI32.CFG 中, 在任何工作目录中都能访问, 并且对任何使用 BDE 的应用程序都是可见的。

下面将通过为子目录 “D:\DelphiApp\14” 建立一个公共数据库别名 App14 来说明数据库别名的创建方法。创建步骤如下。

(1) 在数据库桌面应用程序中执行【Tools】→【Alias Manager】命令, 将会出现如图

14-10 所示的【Alias Manager】(别名管理)对话框。

(2) 单击【New】按钮以创建数据库别名。在【Database alias】列表框中输入数据库别名“APP14”，在【Driver type】列表框中选中“STANDARD”类型，此时将出现一个【Path】文本框，在该文本框中输入路径“D:\DelphiApp\14”，或单击【Browse】按钮，在弹出的【Directory Browser】(浏览目录)对话框中找到所需要的目录“D:\DelphiApp\14”，然后单击【OK】按钮，退出【Directory Browser】对话框。此时的【Alias Manager】对话框如图 14-11 所示。



图 14-10 【Alias Manager】对话框



图 14-11 新建数据库别名的【Alias Manager】对话框

(3) 单击【Alias Manager】对话框上的【OK】按钮，系统将弹出如图 14-12 所示提示信息，询问用户是否将数据库别名保存到 BDE 配置文件 IDAPI32.CFG 中，单击【是】按钮即可。

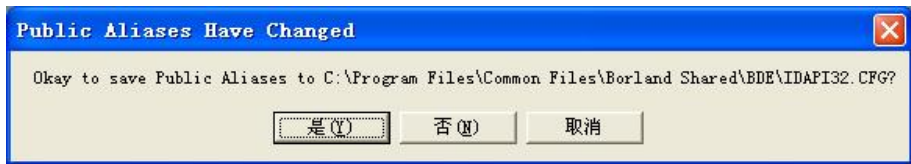


图 14-12 保存数据库别名提示信息

14.1.3 利用 BDE 组件开发数据库应用程序的模式

在 Delphi 中进行数据库应用程序的开发，大部分都可以通过组件来实现。利用 BDE 组件开发数据库应用程序一般要涉及三类组件：BDE 组件、Data Access 组件和 Data Controls 组件，这些组件的关系如图 14-13 所示。

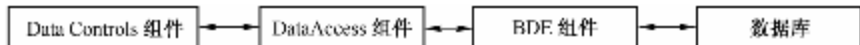


图 14-13 Delphi 各类组件之间的关系

1. BDE 组件

BDE 组件的作用是与实际的物理数据库建立联系，从物理数据库中提取信息，或把信息的修改存放到物理数据库中。该类组件又称数据集组件，都是非可视的组件。最常用的 BDE 组件有 TTable、TQuery 等。BDE 组件直接与实际的物理数据库打交道，从物理数据库中提取信息构成数据集。BDE 组件位于组件板上的 BDE 选项卡中，如图 14-14 所示。

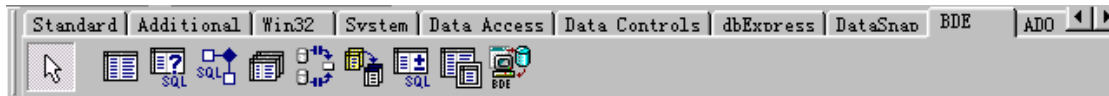


图 14-14 BDE 选项卡

2. Data Access 组件

Data Access 组件位于 Data Access 选项卡中，该类组件是连接 BDE（Borland Database Engine，Borland 数据库引擎）组件和数据控制组件的桥梁，通常又称数据存取组件。该类组件中有一个非常重要的组件——TDataSource 组件，该组件是用户使用最多的组件。在程序运行时，该类组件也是不可见的。

3. Data Controls 组件

Data Controls 组件位于 Data Controls 选项卡中，通过该类组件可以显示和修改数据库中的数据，常把它们称为数据控制组件。与 BDE 组件和 Data Access 组件不同的是，该类组件是可视化组件。常用的数据控制组件有：TDBGrid 组件、TDBEdit 组件、TDBNavigator 等组件等。

14.1.4 TTable 组件

TTable 组件用于和物理数据表建立联系，以便应用程序可以存取数据库中的数据。该组件的属性、方法和事件很多，要掌握该组件功能，应首先掌握它的一些常用事件、属性和方法。

1. TTable 组件的常用属性

(1) DatabaseName 属性

该属性用于指定要连接的物理数据库的名字，可以是数据库别名、数据表文件所在的磁盘路径。通常给该属性赋值一个数据库别名，这样做的好处是当数据库存放的物理位置发生变化时，只需重新使用 BDE 设置数据库别名，而无需改变数据库应用程序。当在属性窗口中选中该属性时，在其后将出现下拉列表，列表中列出了当前系统已经建立的数据库别名，供用户选择。

(2) TableName 属性

该属性用于设定 TTable 组件所操作的数据表名称，通常在 DatabaseName 属性设置后设置。设置了 DatabaseName 属性后，在属性窗口中选中 TableName 属性时，将会在其后出现一个列表框，该列表框中列出了相应数据库中的所有表，供用户选择。DatabaseName 属性和 TableName 属性可以在设计阶段设置也可在程序运行过程中设置，但需要注意的是：要修改这两个属性的值，必须先把 TTable 组件的 Active 属性设置为 False，否则这两个属性的值将

不能被修改。

(3) TableType 属性

该属性用于设定与 TTable 组件相连接的数据库表的类型。当该属性设置成 ttDefault 时，数据库表类型由数据表文件的扩展名决定，扩展名与类型之间的关系如下。

! 扩展名为.db 或没有扩展名，数据表的类型是 Paradox 表。

! 扩展名为.dbf 时，数据表的类型是 dBASE 表。

! 扩展名为.txt 时，表的类型是 ASCII 表。

(4) Active 属性

该属性用于打开或关闭 TTable 组件形成的数据集合，是一个逻辑型属性。值为 True 时，表示数据集是打开的，可以对 TTable 组件连接的数据表中的数据进行操作；值为 False 时，表示数据集是关闭的，此时不可以对 TTable 组件连接的数据表中的数据进行操作。

(5) ReadOnly 属性

该属性是逻辑型属性，用来决定 TTable 组件形成的数据集是否可以修改。值为 True 时，为只读，不能修改，值为 False 时，可以修改。

(6) Modified 属性

该属性是一个只读属性，只能在程序运行中使用。该属性用于表示数据表中的数据是否被修改过。如果值为 True，表示数据已被修改，如果值为 False，则表示数据没有被修改。在程序中，一般可以通过 Modified 属性的值来判断当前数据库中的数据是否被修改，从而决定相应的操作。

(7) Fields 属性

该属性是一个数组属性，它的每一个元素代表着数据集的一个字段，是 Tfield 类型的。Fields 属性又称 Fields 集合。通过该属性可以访问数据集中的字段：Fields[0]代表第一个字段，Fields[1] 代表着第二个字段，依此类推。

注意：引用数据集合中的字段，还可以通过 FieldByName 方法来实现。

(8) Bof 属性

该属性用于指示记录指针是否指向数据集的第一条记录。如果记录指针指向第一条记录，则 Bof 值为 True，否则 Bof 值为 False。该属性是只读属性。

(9) Eof 属性

该属性用于指示记录指针是否指向数据集的最后一条记录。如果记录指针指向最后一条记录，则 Eof 值为 True，否则 Eof 值为 False。该属性是只读属性。

2. TTable 组件的常用方法

(1) FieldByName 方法

该方法用于引用数据集中的某个字段，该方法的函数形式如下：

```
Function FieldByName(Const FieldName:string):Tfield
```

其中，参数 FieldName 表示要引用的字段名，是一个字符型参数。该函数的返回值是一个 Tfield 对象，通过它可以返回指定字段的相关信息。

(2) 记录指针移动方法

TTable 组件是一种数据集组件，数据集中有一个记录指针，它指向的记录称当前记录。

可以通过移动记录指针来改变当前记录，用户一般对当前记录进行操作。记录指针移动的相关方法如表 14-2 所示。

表 14-2 TTable 组件记录指针移动方法

记录指针移动方法	功 能
First 方法	将记录指针移到第一条记录的位置
Last 方法	将记录指针移到最后一条记录的位置
Prior 方法	将记录指针移到上一条记录的位置
Next 方法	将记录指针移到下一条记录的位置
MoveBy 方法	将记录指针从当前记录开始向前或向后移到若干条记录，格式为： Function MoveBy(Distance:Integer):Integer; 参数 Distance 可以为正负整数，如果为正整数则指针向表尾方向移动 Distance 条记录，为负数则表示向表头方向移动 Distance 的绝对值条记录

(3) 查询记录的相关方法

TTable 组件提供了四种查询记录的方法，分别是 GotoKey，GotoNearest，FindKey，FindNearest。其中，GotoKey 和 FindKey 是用于精确查找，GotoNearest 和 FindNearest 用于模糊查找。

使用这四种方法查找记录前，必须保证要查找的字段是关键字或者已经为它建立了辅助索引，并且保证 TTable 组件的属性中已经设置了关键字段名和辅助索引名，可以通过 TTable 组件的 IndexFieldNames 和 IndexName 属性进行选择。

① GotoKey 方法：用于精确查找，使用方法的步骤如下。

- 确保要查找的字段是关键字或已经为它定义了辅助索引，并保证 TTable 组件的属性列表中有关键字段名 (IndexFieldNames) 或辅助索引名 (IndexName)。
- 通过调用 SetKey 方法，把要查找的 TTable 组件置成查找模式。
- 把查找值送进被查找的 Field 的查找缓冲区。
- 调用 TTable 组件的 GotoKey 方法，并测试它的返回值以判断查找是否成功。

该方法执行后，如果查找成功，GotoKey 将返回 True，并且表中的记录指针指向找到的记录。如果查找失败，GotoKey 将返回 False，表中的记录指针不发生变化。

需要注意的是，如何给 Field 的查找缓冲区赋值，由于字段对象是不可见的，因此在大多数情况下，要使用 TTable 组件的 FieldByName 方法到字段列表中查找字段对象以便为它赋值，当 TTable 组件处于查找模式时，只要把查找值赋给字段对象的 AsString 属性就可以了。AsString 的作用不只是它的表面意思，它还是一个转换属性，任何赋给字段对象的 AsString 属性的字符串都将转换成该字段对象对应于数据库表中的字段的数据类型。但 AsString 不能将查找值转换成 BLOB、Bytes、Memo 和 Graphic 类型的数据。

GotoNearest 方法用于不精确查找，是一个过程，没有返回值，它把记录指针定位于最接近于查找值的记录位置上。处理过程基本同 GotoKey。

② FindKey 方法：是实现精确查找的另一种方法，它和 GotoKey 一样均是一个函数过程，其格式如下：

```
function FindKey(const Key Values: array of const): Boolean;
```

与 **GotoKey** 方法的区别在于：**GotoKey** 不带参数，而 **FindKey** 带有参数，其参数就是要索引查找的值。**FindKey** 接受的值是放在方括号中的用逗号分开的数组，数组中的每一个值都对应于特定列的查找值，即参数中允许有多个查找值，该方法允许用户同时查找数据库表中的多个列。

该方法执行后，如果查找成功，将返回 **True**，并且表中的记录指针指向找到的记录。如果查找失败，将返回 **False**，表中的记录指针不发生变化。

FindNearest 方法用于不精确查找，是一个过程，没有返回值，它把记录指针定位于最接近于查找值的记录位置上。处理过程基本同 **FindKey**。

（4）数据操作的相关方法

除实现数据表的浏览、查询外，**TTable** 组件还有一个重要的功能就是进行数据维护，如实现记录的添加、删除和修改等操作。这些数据维护都是通过 **TTable** 组件的数据操作方法来实现的，常用的数据操作方法的名称和功能如表 14-3 所示。

表 14-3 TTable 组件的数据操作方法

数据操作方法名称	功 能 说 明
Append	在数据表的末尾追加一条空记录，并将记录指针指向该记录
Insert	在当前记录之后插入一条空记录，并将记录指针指向该记录
Delete	删除当前记录，并将指针移向下一条记录
Edit	将数据表置于编辑状态，此时用户可以修改数据表中的数据
Post	把对数据表数据的修改结果和新添加的数据保存到数据表中
Cancel	取消对数据表中数据的修改和添加
Refresh	根据数据表中的数据刷新界面，即重新从数据表中获取数据
EmptyTable	将数据表清空，即删除数据表中的所有记录

注意：使用 **Post** 或 **Cancel** 方法之前，必须已经调用过 **Append** 方法或 **Insert** 方法。

3. TTable 组件的常用事件

（1）Before 事件

该类事件在相应的操作发生之前被触发，**Before** 后通常跟相应操作的名称，事件名及其发生时机如表 14-4 所示。

表 14-4 Before 事件的发生时机及作用

事 件	发生时机及作用
BeforeCancel	该事件在调用数据集的 Cancel 方法时，在取消操作发生之前触发，是调用 Cancel 方法发生的第一个动作
BeforeClose	在关闭数据集之前发生，是在数据集关闭之前发生的第一个动作
BeforeDelete	该事件在调用数据集的 Delete 方法时，在删除操作发生之前触发，是调用 Delete 方法发生的第一个动作，通常用来对删除进行确认
BeforeEdit	该事件在调用数据集的 Edit 方法时，在把数据表置为编辑模式之前触发，是调用 Edit 方法发生的第一个动作
BeforeInsert	该事件在调用数据集的 Append 或 Insert 方法时，在插入或添加操作发生之前触发，是调用 Append 或 Insert 方法发生的第一个动作
BeforeOpen	在数据集被打开之前发生的第一个动作
BeforePost	该事件在调用数据集的 Post 方法时，在更新操作发生之前触发，是调用 Post 方法发生的第一个动作

(2) After 事件

该类事件在相应操作完成之后时触发，After 后通常跟相应操作的名称，事件名及其发生时机如表 14-5 所示。

表 14-5 After 事件的发生时机

事 件	发生时机和作用
AfterCancel	在调用 Cancel 方法之后发生
AfterClose	在关闭数据集之后发生
AfterDelete	在数据集调用 Delete 方法删除记录之后发生
AfterEdit	在数据集调用 Edit 方法之后发生
AfterInsert	在数据集对 Insert 或 Append 方法调用之后发生
AfterOpen	在数据集被打开之后发生
AfterPost	在数据集调用 Post 方法保存数据之后发生

该类事件通常用于在某类操作完成过后给出提示信息。

(3) On 事件

On 事件在相应操作发生时发生，On 后通常跟相应操作的名称，事件名及其发生时机如表 14-6 所示。

表 14-6 On 事件的发生时机

事 件	发生时机和作用
OnNewRecord	在添加一个新的记录时被触发。该事件在 BeforeInsert 事件之后发生，在 AfterInsert 事件之前发生
OnFilterRecord	该事件当数据集过滤器处于激活状态（Filter 属性为 True）并且将记录指针移到其他记录时发生

14.1.5 TDataSource 组件

TDataSource 组件是使用最多的数据访问组件，用于连接数据集组件和数据控制组件。

TDataSource 组件最重要的属性是 DataSet 属性。该属性指出要与哪一个数据集组件建立联系，它的属性值是某一个数据集组件的对象名称，如：TTable、TQuery 等组件的名称。TDataSource 组件通过 DataSet 属性与数据集组件建立连接后，数据控制组件就可以借助 TDataSource 组件与数据集组件（如 DBEdit 和 DBGrid 等组件）建立连接，这样就可以成功地访问数据库中的数据了。

14.1.6 Data Controls 组件

在 Delphi 的组件板中有一个【Data Controls】选项卡，在该选项卡中集中了一些常用的数据控制组件，如图 14-15 所示。数据控制组件的名称前面都带有 DB，大部分功能均类似于 Delphi 相应的标准组件，只不过它和数据库表或数据库表的字段绑定在一起而已。

1. TDBNavigator 组件

TDBNavigator 组件的形状及其上按钮的名称如图 14-16 所示。

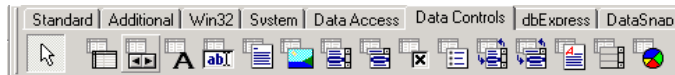


图 14-15 Data Controls 选项卡中的数据控制组件

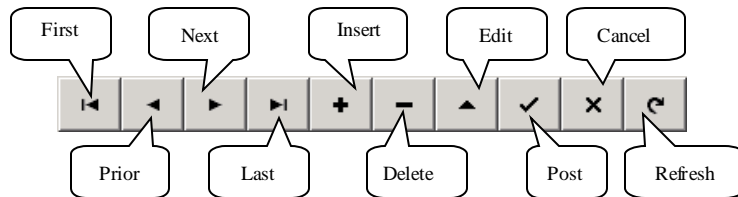



图 14-16 TDBNavigator 组件及其按钮名称

TDBNavigator 组件中各种按钮的功能如表 14-7 所示。

表 14-7 TDBNavigator 组件各按钮的功能

按钮名称	按钮功能
First	将数据集的记录指针移到第一条记录
Prior	将数据集的记录指针移到前一条记录
Next	将数据集的记录指针移到后一条记录
Last	将数据集的记录指针移到最后一条记录
Insert	在当前记录之后插入一条记录
Delete	删除当前记录，当 ConfirmDelete 属性设置为 true 时，会弹出删除确认对话框
Edit	使数据集处于编辑状态
Post	把对数据集的修改（Insert 或 Edit），保存到实际的数据表中
Cancel	取消对数据集的修改
Refresh	根据数据表的内容，刷新数据组件中的数据

2. TDBGrid 组件

TDBGrid 组件是用于显示和编辑数据库表中记录信息的重要组件。该组件以列表的形式来显示数据库中的数据记录，它包含了数据的行和列。在网格中每一行代表数据集中的一行，每一列代表数据集中的—个字段。它的用法很简单，是最常用的数据控制组件之一。组件图标为.

3. TDBListBox 组件 TDBComboBox 组件

TDBListBox 组件 TDBComboBox 组件类似于 TListBox 组件和 TComboBox 组件，TDBListBox 组件用于在一个列表中显示或设置字段的值，TDBComboBox 用于在一个文本框中显示或设置字段的值。TDBListBox 只能从列表框中选择数据更新数据表的数据，而 TDBComboBox 既可以在组合框的列表中选择—个值也可以自行输入数据来更新数据表的数据。当某个字段只有有限个值时，为了简化输入和避免输入失误，可选用这两个组件。

6. TDBImage 组件

该组件用于显示图像字段内容。它有三个重要属性：DataSource、DataField 和 Stretch。其中 DataField 应和—个图像字段相关联。Stretch 属性是—个逻辑型属性，当它的值为 True 时，要显示的图像将会自动调整大小以适应 TDBImage 组件的大小，而且当 TDBImage 组件

的尺寸改变时, 图像的尺寸也跟着改变。

7. TDBCheckBox 组件

该组件有两个重要的属性: `ValueChecked` 和 `ValueUnchecked`。

■ `ValueChecked` 属性: 该属性是字符串属性, 当字段的内容与该属性值匹配时, 该组件被选中。

■ `ValueUnchecked` 属性: 该属性也是字符串属性, 当字段的内容与该属性值匹配时, 该组件被清除。

如果选用字段既不能和 `ValueChecked` 属性相匹配, 也不能和 `ValueUnChecked` 属性相匹配, 则 `TDBCheckBox` 会呈灰色状态。

如果选用字段本身是逻辑型的, 则当选用字段为 `True` 时, `TDBCheckBox` 就会呈现选中状态, 而选用字段为 `False` 时, `TDBCheckBox` 就会呈现为未选中状态。

8. TDBRadioGroup 组件

该组件主要用于为单选按钮分组。当组中的一个单选按钮被选中时, 同组的其他所有单选按钮均处于未被选中状态。经常使用的该组件的属性有三个: `Columns`、`Items` 和 `Values`。

(1) `Columns` 属性: 用于设置排列单选按钮的列数, 默认值为 1, 表示以一列的形式显示单选按钮。用户可以设置成多列排列方式。

(2) `Items` 属性: 用于设置单选按钮显示的文字。选中该属性, 单击其后的【...】按钮将会出现【String List Editor】对话框, 用户在其中可以每行书写一个单选按钮的提示文字(有时也称单选按钮的名称)。

(3) `Values` 属性: 也是一个字符串列表, 设置方法与 `Items` 属性一样。表示相应单选按钮的值。

`TDBRadioGroup` 组件可确保用户为字段输入一个值, 也可用来显示取值有限的字段。如果用户选中 `TDBRadioGroup` 中的一个单选按钮, 它的 `Value` 属性的值就会传回底层数据库。`Value` 属性和 `Items` 属性类型一样, 都是字符串列表, 而且其中的字符串是一一对应的。

【例 14-1】 编写一个通讯录维护程序, 程序的设计界面如图 14-17 所示, 程序运行时单击 `TDBNavigator` 组件的相应按钮将对通讯录表实现相应的操作, 程序运行界面如图 14-18 所示。

图 14-17 程序设计界面

图 14-18 程序运行界面

【实现分析】

可使用 TTable 组件和数据控制组件来维护和操作数据库中的内容,使用 TTable 组件和数据控制组件维护与操作数据库中数据的方法步骤如下。

(1) 在界面上添加一个 TTable 组件,通过设置它的 DatabaseName 和 TableName 属性使它与其某一个数据表联系在一起,然后把它的 Active 属性设置为 True;

(2) 在界面上添加一个 TDataSource 组件,设置它的 DataSet 属性值为步骤(1)中添加的 TTable 组件的名称,以便能够通过它访问数据库中的数据;

(3) 在界面上添加数据控制组件,设置它的 DataSource 属性值为步骤(2)中添加的 TDataSource 组件的名称,设置它的 DataField 属性为要显示和设置的字段名称。

【界面设计】

根据图 14-17 所示为窗体添加 TLabel 组件并设置它们的 Caption 属性,以显示提示文字。其他组件对象及其属性设置如表 14-8 所示。

表 14-8 例 14-1 的主要组件对象及其属性设置

对 象 名	属 性 名	属 性 值	说 明
Table1	DatabaseName	'D:\DelphiApp\14\A\A_14_1'	用来与通讯录表相联系
	TableName	'通讯录'	
	Active	True	
DataSource1	DataSet	'Table1'	联系 Table1 组件与数据控制组件
DBEdit1	DataSource DataField	'DataSource1' '好友号'	显示和修改“好友号”字段内容
DBEdit2	DataSource DataField	'DataSource1' '姓名'	显示和修改“姓名”字段内容
DBEdit3	DataSource DataField	'DataSource1' '好友类别'	显示和修改“好友类别”字段内容
DBEdit4	DataSource DataField	'DataSource1' '年龄'	显示和修改“年龄”字段内容
DBEdit5	DataSource DataField	'DataSource1' '出生日期'	显示和修改“出生日期”字段内容
DBEdit6	DataSource DataField	'DataSource1' '电话号码'	显示和修改“电话号码”字段内容
DBEdit7	DataSource DataField	'DataSource1' '通讯地址'	显示和修改“通讯地址”字段内容
DBEdit8	DataSource DataField	'DataSource1' '电子邮件'	显示和修改“电子邮件”字段内容
DBEdit9	DataSource DataField	'DataSource1' 'QQ 号码'	显示和修改“QQ 号码”字段内容
DBMemo1	DataSource DataField	'DataSource1' '交往大事'	显示和修改“交往大事”字段内容
DBNavigator1	DataSource	'DataSource1'	浏览和维护“通讯录”表的内容

【程序代码】

直接运行程序即可实现需要的功能,但当把数据库和程序复制到其他位置,执行时将找不到数据库。为了使程序通用,可先把 Table1 的 Active 属性设置为 False,然后在 FormCreate

事件中添加如下事件代码:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:string;
begin
    Getdir(0,CurDir);           //获得应用程序所在路径
    Table1.DatabaseName :=CurDir; //把应用程序所在路径赋值给 Table1 的 DatabaseName
    Table1.open;               //打开数据库
end;
```

14.1.7 SQL 语言

Delphi 中提供了对 SQL 语言的支持。对一般用户而言,在 Delphi 中使用的 SQL 语言的功能主要有:数据查询、插入记录、删除记录、更新记录等。

1. 数据查询

数据查询是数据库中最常见的操作,SQL 中实现查询的语句是 SELECT 语句,该语句由一系列的子句组成,通过这些子句可以实现对数据库的任意查询。该语句的格式和功能如下。

[格式]:

```
SELECT [ALL | DISTINCT |TOP N|TOP N PERCENT]
    *|列名 1 或表达式 1  [AS  列标题 1 ][,列名 2 或表达式 2  [AS  列标题 2]...]
FROM  表名 1 [IN 数据库名 1] 别名 1[,表名 2 [IN 数据库名 2] 别名 2 ...]
[WHERE  条件]
[GROUP BY 列名 1 [, 列名 2]...]
[HAVING 条件]
[ORDER BY 列名 1 [ASC | DESC] [, 列名 2 [ASC | DESC]...]]
```

[功能]:从“表名 1”和“表名 2”等指定的表中查询满足条件的数据。

[说明]:

(1) ALL | DISTINCT | TOP N | TOP N PERCENT: ALL 表示值相同的记录也包含在结果中,这是默认设置;DISTINCT 表示对于相同的记录只包含第一条记录;TOP N 只返回查询结果的前 N 条记录,N 为可变参数;TOP N PERCENT 表示只返回查询结果中的前百分之 N 条记录,N 为可变参数。

(2) *|列名 1 或表达式 1 [AS 列标题 1][,列名 2 或表达式 2 [AS 列标题 2]...]:用于指定查询结果中包含的选定项,可以是列名或表达式,AS 后面的“列标题”用来定义对应列或表达式的标题。在“列名 1”、“列名 2”等别名的前面可能有包含该列的表名。若选“*”表示输出列是由 FROM 子句中指定的表中的所有字段。

(3) FROM 表名 1 [IN 数据库名 1] 别名 1[,表名 2 [IN 数据库名 2] 别名 2 ...]:用于指定查询操作从哪些表中获取数据,每个表可以取一个“别名”,这样就可以在选定项中用“别名”来引用相应表中的字段。

(4) WHERE 条件:用于设定查询条件或多个表之间连接条件。

(5) **GROUP BY** 列名 1 [, <列名 2>...]: 用于指定按照哪些列进行分组, 以便进行统计。

(6) **HAVING** 条件: 只有在指定分组的查询中才有效, 用来对分组的结果进行筛选, 只有满足条件的组信息才包含在查询结果中。

(7) **ORDER BY** 列名 1 [**ASC** | **DESC**] [, 列名 2 [**ASC** | **DESC**]...]: 用于指定按照哪些列对查询结果进行排序, **ASC** 代表升序, **DESC** 代表降序, 默认为升序。

(8) 格式中的所有条件均可以是逻辑表达式或关系表达式, 使用的运算符有: **AND**、**OR**、**NOT**、**=**、**<**、**<=**、**>**、**>=**、**<>**、**BETWEEN**、**LIKE**、**IN** 等。

例如, 如下语句:

```
Select 学号,姓名,年级 from 学生 Where 专业="自动化"
```

其作用是列出自动化专业的全部学生的学号、姓名和年级。

又如:

```
SELECT 学生.学号,学生.姓名,必修课成绩.课号,必修课成绩.成绩  
FROM 学生,必修课成绩 WHERE 学生.学号=必修课成绩.学号
```

其作用是查询出所有学生的必修课的学习情况, 查询结果中包含学号、姓名、课号和成绩。

2. 插入记录

插入记录可使用 SQL 语言的 **INSERT** 语句, 该语句的格式和功能如下。

[格式]:

```
INSERT INTO 表名[(字段名 1 [, 字段名 2, ...])]  
VALUES (表达式 1 [,表达式 2,...])
```

[功能]: 向“表名”指定的表中插入一条记录, **VALUES** 后面括号中的数据就是新记录的相应字段的数据。即“表达式 1”的值作为“字段名 1”中的数据, “表达式 2”作为“字段名 2”中的数据, ……如果要给记录的所有字段插入值, 表名后面的字段名可以缺省, 但插入数据的类型必须与表中相应字段的数据类型完全吻合; 若只需要插入表中某些字段的数据, 就需要列出要插入数据的字段名, 并给出相应的值。

例如, 下列语句:

```
INSERT INTO 学生(学号,姓名,专业) VALUES("040501","朱碧春","计算机软件")
```

其功能是向学生表中插入一条记录, 并给学号、姓名和专业字段赋值。

3. 修改记录

在 SQL 语言中, 对存放在表中的数据进行修改可以通过 **UPDATE** 语句来实现。UPDATE 语句的格式和功能如下。

[格式]:

```
UPDATE 表名 SET 字段名 1 = 表达式 1[,字段名 2=表达式 2...][WHERE 条件]
```

[功能]: 对由“表名”指定的表中满足 **WHERE** 后面的“条件”的记录(若无 **WHERE** 子句将修改所有记录)进行修改, 修改方法是: 以“表达式 1”的值替换“字段名 1”中的值, 以“表达式 2”的值替换“字段名 2”中的值, 以此类推。

例如, 如下语句:

```
UPDATE 必修课成绩 SET 成绩=成绩+3 WHERE 课号="02"
```

其作用是将“必修课成绩”表中的“02”号课的所有成绩增加3分。

4. 删除记录

当表中的记录已经失效时,应把它删除,可以使用 SQL 的 DELETE 命令来删除表中的记录。DELETE 命令的格式和功能如下。

[格式]:

```
DELETE FROM 表名 [WHERE 条件]
```

[功能]: 从由“表名”指定的表中删除满足给定“条件”的记录,当缺省 WHERE 子句时,表示删除表中的所有记录。

例如,如下语句:

```
DELETE FROM 学生 WHERE 专业="经济管理"
```

其作用是从“学生”表中删除所有的“经济管理”专业的学生。

14.1.8 TQuery 组件

TQuery 组件也是一种 BDE 组件,其在数据库应用程序中所处的位置与 TTable 组件基本一致,与 TTable 组件不同的是,TTable 组件与一个具体的数据表相联系,而 TQuery 组件并不和某一个具体的数据表联系在一起,它主要用于执行 SQL 语句,通过 SQL 语句可以操作一个或多个表。

1. TQuery 组件的常用属性

(1) DataBaseName 属性

该属性用于与数据库建立连接,可以是某个数据库别名或数据表所在的路径。其设置和使用方法与 TTable 组件相同。

(2) SQL 属性

与 TTable 组件不同,TQuery 组件是通过执行 SQL 语句来对数据表进行操作的,SQL 属性中存放的就是 TQuery 组件要执行的 SQL 语句。该属性是一个字符串属性,字符串中包含的是一条 SQL 语句,如 SELECT、INSERT、UPDATE、DELETE 语句等。在 SQL 属性中输入了合法的 SQL 语句后,可以通过将它的 Active 属性设置为 True 或者调用它 Open 方法(或 ExecSQL)来执行该 SQL 语句。另外,该属性还提供了设置 SQL 语句的一些方法,如 Add、Clear 等,这些方法将在后面进行介绍。

(3) Active 属性

当该属性设置为 True 时,将执行 SQL 属性中存放的 SQL 语句。

2. TQuery 组件的常用方法

TQuery 组件也有很多方法,许多方法与 TTable 组件基本一致,如控制记录指针移动的方法有: First、Next、Prior 和 Last 等。这些方法可参考 TTable 组件的相关内容。

(1) Open 方法

该方法用于打开或激活 TQuery 组件的数据集。使用该方法,TQuery 组件的 SQL 属性中存放的应是 Select 查询语句,其结果产生一个数据集。Open 方法只能执行 Select 语句,结果

返回一个数据集。

(2) ExecSQL 方法

该方法也是执行 TQuery 组件中的 SQL 语句。与 Open 方法不同的是,该方法不仅能执行查询 (SELECT) 语句,而且能够执行添加 (INSERT)、删除 (DELETE) 和更新 (UPDATE) 的 SQL 语句。

一般情况下,如果要执行的 SQL 命令是一条 SELECT 语句,使用 Open 方法,但如果要执行的 SQL 命令是添加、删除或者更新的语句,则必须调用 ExecSQL 方法执行。

(3) Close 方法

该方法用来关闭数据集。

(4) Prepare 方法

调用该方法,Delphi 会将带参数的 SQL 语句传送给对应的数据库引擎,对 SQL 语句进行语法分析和优化,从而大大提高动态 SQL 语句的执行性能,特别是当反复多次执行同一条动态 SQL 语句时,它的优越性会更加明显。

(5) SQL 属性的两个基本方法

① Add 方法:用于为 TQuery 组件的 SQL 属性添加 SQL 语句,该方法的语法格式如下:

```
Query.SQL.Add(SQLState);
```

其中,Query 是 TQuery 组件的名称,参数 SQLState 是一个字符串,字符串中存放的是 SQL 语句,例如:

```
Query1.SQL.Add('Select * From 通讯录')  
Query1.SQL.Add(' Where 好友类别="朋友"');    //在前面 SQL 语句基础上追加 SQL 语句
```

② Clear 方法:用于清除 TQuery 组件的 SQL 属性值,一般在用 Add 方法添加 SQL 语句前调用该方法将 SQL 属性清空,然后再添加全新的 SQL 语句。例如:

```
Query1.Close;           //先关闭数据集  
Query1.SQL.Clear;       //将 SQL 属性值清空  
Query1.SQL.Add('Select * From 通讯录 Where 姓名="李建东"');  
//添加 SQL 语句,在字符串里面用两个单引号表示一个单引号。  
Query1.Open;           //打开数据集,即执行 SQL 语句
```

3. 使用 TQuery 组件实现参数查询的方法

(1) 使用 Params 属性实现参数查询

TQuery 组件具有一个 Params 属性,该属性只能在程序运行时访问,在程序设计时不可以使用,并且该属性是动态建立的数组属性,它的每一个元素对应要执行的 SQL 语句中一个参数。在 SQL 语句中可以使用参数,参数的表示形式是以“:”开头的一个标识符。数组属性 Params 是以 0 下标开始的,依次对应动态 SQL 语句中的参数,即 Params[0]对应第一个参数,Params[1]对应第二个参数,依此类推。如下列语句:

```
Query1.SQL.Add('Select * From 通讯录');  
Query1.SQL.Add('Where 姓名=:XM');           //注意参数的前面有":"号  
Query1.Params[0].AsString := '李建东';      //给参数赋值
```

```
Query1.Open;
```

```
//打开查询结果数据集
```

Query1 的 SQL 属性中的 SQL 语句中有一个参数, 参数名为 XM。程序执行时, 系统将自动创建 Query1 组件的 Params 数组属性, 该数组将有一个元素 Params[0], 对应参数 XM。在执行该 SQL 语句之前必须给相应的参数赋值, 如语句 “Query1.Params[0].AsString := '李建东'” 就是给参数 XM 赋值为 '李建东'。

(2) 使用 ParamByName 方法实现参数查询

除了可以使用 TQuery 组件的 Params 数组属性给参数赋值外, 还可以使用 TQuery 组件的 ParamByName 方法为参数赋值。ParamByName 方法是一个函数, 其参数是一个动态 SQL 语句的参数名, 故使用该语句必须要知道动态 SQL 语句参数的名字。

如上述的语句也可以写成:

```
Query1.SQL.Add('Select * From 通讯录');
```

```
Query1.SQL.Add('Where 姓名=:XM');
```

```
//注意参数的前面有":"号
```

```
Query1.ParamsByName("XM").AsString := '李建东';
```

```
//给参数赋值
```

```
Query1.Open;
```

```
//打开查询结果数据集
```

【例 14-2】 给定出生日期范围, 查询通讯录表中出生日期在该范围内的好友的通讯录信息。程序的设计界面如图 14-19 所示, 程序的运行界面如图 14-20 所示。程序运行时在两个文本框中输入起始日期, 然后单击【查询】按钮将把出生日期在该范围之内的好友的通讯录信息显示在网格组件中, 如图 14-20 所示。



图 14-19 程序设计界面



图 14-20 程序运行界面

【实现分析】

本例需根据用户输入的数据来查询指定的信息, 可通过使用 TQuery 组件执行 SELECT 查询语句来实现。可使用参数查询, 设定两个参数, 分别用来接收用户输入的信息, 然后可通过 TQuery 组件的 Params 属性或 ParamByName 方法给参数赋值, 再执行查询得到结果。

【界面设计】

本例组件对象及其属性设置如表 14-9 所示。

表 14-9 例 14-2 的组件对象及其属性设置

对象名	属性名	属性值	说明
Query1			执行查询
DataSource1	DataSet	'Query1'	联系 Query1 组件和 DBGrid1 组件

续表

对 象 名	属 性 名	属 性 值	说 明
DBGrid1	DataSource	'DataSource1'	显示查询结果
Label1	Caption	'开始日期'	
Edit1	Text	"	输入起始日期
Label2	Caption	'结束日期'	
Edit2	Text	"	输入结束日期
Button1	Caption	'查询'	单击它将执行查询

【程序代码】

```
procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:string;
begin
    Getdir(0,CurDir);
    Query1.DatabaseName :=CurDir;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Query1.SQL.Clear ;                               //清除 SQL 属性中原来的语句
    Query1.SQL.Add('Select * From 通讯录');
                                //以下两条语句用来给 SQL 属性添加新的 SQL 语句
    Query1.SQL.Add('Where 出生日期>=:D1 AND 出生日期<=:D2'); //两个参数 D1 和 D2
    Query1.Params[0].AsDate :=strtodate(Edit1.Text) ;       //通过 Params 属性给参数赋值
    Query1.ParamByName('D2').AsDate :=StrToDate(Edit2.Text) ;
                                //通过 ParamByName 方法给属性赋值
    Query1.Prepare ;                                       //优化
    Query1.Open ;                                         //执行 SQL 语句，打开记录集
end;
```

14.2 典型实例

14.2.1 典型实例一

【实例题目】：通讯录浏览与维护程序（使用数据控制组件）

建立一个应用程序，用来对本章建立的“通讯录”表进行维护和管理，该程序能够实现数据库日常操作功能，这些功能包括：通讯录数据的浏览、添加一条新的通讯信息、修改通讯信息、查询某人的通讯信息、删除某人的通讯信息等。程序的主界面设计时和运行时的情况分别如图 14-21 和图 14-22 所示。要求显示数据库中的数据采用 Data Controls 组件板上的相应组件，查询采用 SetKey 方法按姓名进行查询。



图 14-21 程序设计界面

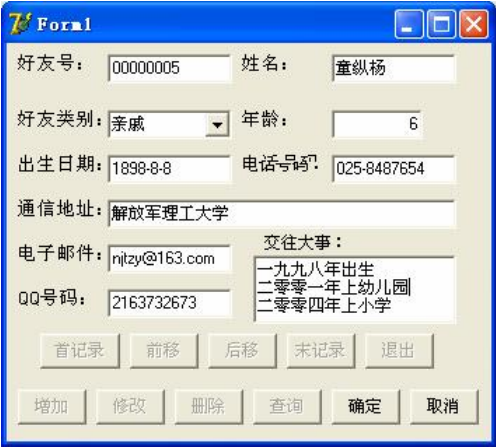


图 14-22 程序运行界面（修改时）

【实现方法】

(1) 功能按钮的禁止使用和允许使用。在一定的情况下，有的按钮不可以使用，如记录指针移到数据库开头时【首记录】和【前移】按钮应不可使用。如果数据表中无记录，则除【添加】按钮外的其他按钮均不可用。一开始，【确定】和【取消】按钮不可用，单击【添加】或【修改】按钮后，【确定】和【取消】按钮可用，此时其他各按钮不可用。当单击【确定】和【取消】按钮后，这两个按钮不可用，其他各个按钮可用。当记录指针处于文件尾时，【后移】和【末记录】按钮不能用，当记录指针处于文件头时【前移】和【首记录】按钮不能用。

(2) 记录移动。可使用 TTable 控件的相应移动方法。

(3) 记录的添加与修改。对表的操作，TTable 组件提供了相应的方法：如添加一条记录可执行 Append 方法，修改记录的内容可执行 Edit 方法，把修改的内容存入到表中可执行 Post 方法，取消对表的修改可执行 Cancel 方法等。为控制数据只能在“添加”或“修改”时才能输入，可在没有执行“添加”或“修改”命令之前，让所有的数据控制组件不能输入或修改，在执行“添加”或“修改”命令之后，让所有的数据控制组件能够输入或修改。该功能可通过设置各数据控制组件的 ReadOnly 或 Enabled 属性来实现。

【界面设计】

在例 14-1 中去掉 DBNavigator1 组件，用 DBComboBox1 取代 DBEdit3 用来显示好友类别，然后在界面上添加 11 个按钮，用来实现对通讯录表的浏览和维护功能。添加组件的属性设置如表 14-10 所示。

表 14-10 组件的属性设置及组件作用

对象名	属性名	设置值	对象作用
DBComboBox1	DataSource	'DataSource1'	显示和修改“好友类别”字段内容
	DataField	'好友类别'	
	Items	'同学' '朋友' '亲戚'	
Button1	Caption	'首记录'	单击它将把记录指针移到首记录
Button2	Caption	'前移'	单击它将把记录指针前移一条记录

续表

对 象 名	属 性 名	设 置 值	对 象 作 用
Button3	Caption	'后移'	单击它将把记录指针后移一条记录
Button4	Caption	'未记录'	单击它将把记录指针移到未记录
Button5	Caption	'退出'	单击它将退出应用程序
Button6	Caption	'增加'	单击它将增加一条新记录
Button7	Caption	'修改'	单击它将当前记录进行修改
Button8	Caption	'删除'	单击它将当前记录删除
Button9	Caption	'查询'	单击它将按姓名进行查询
Button10	Caption	'确定'	单击它将保存对记录的添加或修改
Button11	Caption	'取消'	单击它将取消对记录的添加或修改

【程序代码】

```

implementation
var
    BM:TBookMark;           //该变量用于在添加记录时把当前记录位置作为一个书签
    AddOrNot:Boolean;       //该变量用于表示是添加操作还是修改操作，True 表示添加
{$R *.dfm}
Procedure ButtonAddEdit();  //用于设置执行添加或修改操作时各按钮的状态
begin
    Form1.Button1.Enabled :=False;    Form1.Button2.Enabled:=False;
    Form1.Button3.Enabled :=False;    Form1.Button4.Enabled :=False;
    Form1.Button5.Enabled :=False;    Form1.Button6.Enabled :=False;
    Form1.Button7.Enabled :=False;    Form1.Button8.Enabled :=False;
    Form1.Button9.Enabled :=False;
    Form1.Button10.Enabled :=True;    // 确定按钮可以使用
    Form1.Button11.Enabled :=True;    //取消按钮可以使用
end;
procedure ButtonOkCancel();    //用来设置执行取消或确定操作时各按钮的状态
begin
    Form1.Button1.Enabled :=True;    Form1.Button2.Enabled:=True;
    Form1.Button3.Enabled :=True;    Form1.Button4.Enabled :=True;
    Form1.Button5.Enabled :=True;    Form1.Button6.Enabled :=True;
    Form1.Button7.Enabled :=True;    Form1.Button8.Enabled :=True;
    Form1.Button9.Enabled :=True;
    Form1.Button10.Enabled :=False;   // 确定按钮不可以使用
    Form1.Button11.Enabled :=False;   //取消按钮不可以使用
end;
procedure Cannedit();          //用于设置各输入的数据控制组件不可输入
begin
    Form1.dbEdit1.ReadOnly:=True;    Form1.dbEdit2.ReadOnly:=True;
    Form1.dbComboBox1.ReadOnly :=True; Form1.dbEdit4.ReadOnly:=True;
    Form1.dbEdit5.ReadOnly:=True;    Form1.dbEdit6.ReadOnly:=True;
    Form1.dbEdit7.ReadOnly:=True;    Form1.dbEdit8.ReadOnly:=True;
    Form1.dbEdit9.ReadOnly :=True;    Form1.dbMemo1.ReadOnly:=True;

```

```

end;
procedure Canedit();           //用来设置各输入的数据控制组件可以输入
begin
    Form1.dbEdit1.ReadOnly:=False;  Form1.dbEdit2.ReadOnly:=False;
    Form1.dbComboBox1.ReadOnly:=False;  Form1.dbEdit4.ReadOnly:=False;
    Form1.dbEdit5.ReadOnly:=False;  Form1.dbEdit6.ReadOnly:=False;
    Form1.dbEdit7.ReadOnly:=False;  Form1.dbEdit8.ReadOnly:=False;
    Form1.dbEdit9.ReadOnly:=False;  Form1.dbMemo1.ReadOnly:=False;
end;
procedure DisabledButton();    //使所有的按钮均不可用
begin
    Form1.button1.Enabled:=False;    Form1.button2.Enabled:=False;
    Form1.button3.Enabled:=False;    Form1.button4.Enabled:=False;
    Form1.button5.Enabled:=False;    Form1.button6.Enabled:=False;
    Form1.button7.Enabled:=False;    Form1.button8.Enabled:=False;
    Form1.button9.Enabled:=False;    Form1.button10.Enabled:=False;
    Form1.button11.Enabled:=False;
end;
procedure TForm1.FormCreate(Sender: TObject);
var
    Curdir:string;
begin
    getdir(0,Curdir);
    Table1.DatabaseName:=Curdir;    //设置 Table1 的 Databasename 为当前目录
    Table1.Active:=True;             //打开 Table1
    CannotEdit;                      //设置可输入组件为只读
    if table1.RecordCount=0 then      //如果数据库表为空
    begin
        disabledButton;
        Button6.Enabled:=True;       //只有添加按钮可用
    end
    else
        ButtonOkCancel;
    Table1.IndexName:='XM';           //设置查询用的索引名
end;
procedure TForm1.Button1Click(Sender: TObject); //首记录
begin
    Table1.First ;
    Button1.Enabled:=False; Button2.Enabled:=False;
    Button3.Enabled:=True; Button4.Enabled:=True;
end;
procedure TForm1.Button2Click(Sender: TObject); //前移
begin
    Table1.Prior ;
    button3.Enabled:=True;  Button4.Enabled:=True;
    if Table1.Bof then      //判断是否到达文件头

```

```

begin
    Table1.First ;
    button1.Enabled :=False;    Button2.Enabled :=False;
end;
end;
procedure TForm1.Button3Click(Sender: TObject);    //后移
begin
    Table1.Next ;
    Button1.Enabled :=True; Button2.Enabled :=True;
    if Table1.Eof then                                //判断是否到达文件尾
    begin
        Button3.Enabled :=False;    Button4.Enabled :=False;
        Table1.Last ;
    end;
end;
procedure TForm1.Button4Click(Sender: TObject); //末记录
begin
    Table1.Last ;
    Button1.Enabled :=True; Button2.Enabled :=True;
    Button3.Enabled :=False; Button4.Enabled:=False;
end;
procedure TForm1.Button5Click(Sender: TObject);    //退出
begin
    Table1.Close ;                                //关闭表
    Application.Terminate ;                        //结束程序运行
end;
procedure TForm1.Button6Click(Sender: TObject);    //添加
begin
    BM:=table1.GetBookmark ;                        //记下当前记录位置
    CanEdit;                                        //允许输入
    Table1.Append ;                                //增加记录
    ButtonAddEdit;                                  //设置按钮状态
    AddOrNot:=True;                                //是增加操作
end;
procedure TForm1.Button7Click(Sender: TObject);    //修改
begin
    AddOrNot:=False;                                //不是添加操作
    Table1.Edit ;                                    //修改
    CanEdit;                                        //让所有的数据控制组件可以使用
    ButtonAddEdit;
end;
procedure TForm1.Button8Click(Sender: TObject);    //删除
begin
    if Application.MessageBox('真的要删除吗? ','删除提示框',MB_OKCANCEL)=IDOK Then
        Table1.Delete ;
        if Table1.RecordCount =0 then                //如果所有的记录均被删除

```

```

        begin
            ShowMessage('已无记录');           //显示提示信息
            DisabledButton;                     //让所有按钮均不可用
            Button6.Enabled :=True;            //再让添加按钮可用，只能执行添加操作
        end;
    end;
    procedure TForm1.Button9Click(Sender: TObject); //按姓名查询
    var
        xm: string;
    begin
        xm:=trim(inputbox('输入姓名','请输入姓名,')); //输入要查询的人的姓名
        with Table1 do
            begin
                setkey;                           //设置查询状态
                FieldByName('姓名').AsString :=xm; //设置要查询的姓名
                if not GotoKey then                 //如果没有找到
                    ShowMessage('无符合条件的记录');
            end
        end;
    end;
    procedure TForm1.Button10Click(Sender: TObject); //确定操作
    begin
        if (length(trim(dbEdit1.text))=0) then //如果“好友号”为空
            begin
                ShowMessage('必须输入好友号');
                dbEdit1.SetFocus ;
            end
        else
            begin
                Table1.Post ;                     //存入到数据库中
                CannotEdit;
                ButtonOkCancel;
            end;
        end;
    end;
    procedure TForm1.Button11Click(Sender: TObject); //取消操作
    begin
        Table1.Cancel;                           //执行取消操作
        if AddOrNot then                          //如果刚才执行的是增加操作
            Table1.GotoBookmark(Bm);              //把指针移到原来的位置
        CannotEdit;                             //不允许输入
        ButtonOkCancel;                          //设置按钮的状态
    end;
end;

```

14.2.2 典型实例二

【实例题目】：SQL 语句执行演示程序

编写一个 SQL 语言的语句执行演示程序, 程序的设计界面如图 14-23 所示, 程序的运行界面如图 14-24 所示。程序操作的数据库为“通讯录”数据库。程序执行时, 选中【Select 语句】单选钮, 可在 Memo1 组件中输入 Select 语句, 输入完毕后, 单击【执行】按钮, 查询的结果将显示在图上面的 DBGrid1 组件中。如果选中【其他数据操纵语句】单选钮, 将在 DBGrid1 中显示“通讯录”表中的信息, 然后可在 Memo1 组件中输入 DELETE、UPDATE、INSERT 等 SQL 语句, 输入完毕后单击【执行】按钮, 将把执行的结果立即显示在 DBGrid1 中。单击【清空】按钮, 将把 Memo1 中输入的内容清空, 单击【退出】按钮将退出应用程序。



图 14-23 程序设计界面



图 14-24 程序运行界面

【实现方法】

使用 TQuery 组件可以执行任意的 SQL 语句, 如果执行的是 SELECT 语句, 一般使用 Open 方法, 并将得到一个结果数据集, 故只需把数据集直接显示在 TDBGrid 组件中即可。如果执行的是其他数据操纵语句, 并不形成数据集, 执行的结果在表中表现出来, 故需要把表中的信息显示出来, 以观看结果。

【界面设计】

按照表 14-11 所示为窗体添加组件并设置组件的属性。

表 14-11 组件的属性设置及组件作用

对象名	属性名	设置值	对象作用
Table1			与“通讯录”表相联系
Query1			执行 SQL 语句
DataSource1			把 Table1 或 Query1 组件与 DBGrid1 组件联系起来
DBGrid1	DataSource	'DataSource1'	显示 SQL 语句执行的结果
GroupBox1	Caption	'SQL 语句种类'	容纳单选钮
RadioButton1	Caption	'Select 语句'	选中它表示要执行的是 SELECT 语句
RadioButton2	Caption	'其他数据操作语句'	选中它表示要执行的是 SQL 的数据操作语句
Memo1			用来输入要执行的 SQL 语句
Button1	Caption	'执行'	单击它将执行输入的 SQL 语句

续表

对 象 名	属 性 名	设 置 值	对 象 作 用
Button2	Caption	'清空'	单击它将把 Memo1 中输入的内容清空
Button3	Caption	'退出'	单击它将退出应用程序

【程序代码】

```

procedure TForm1.FormCreate(Sender: TObject);
var
    CurDir:String;
begin
    Getdir(0,CurDir);
    Table1.DatabaseName :=CurDir;
    Table1.TableName :='通讯录';
    Query1.DatabaseName :=CurDir;
    Memo1.Clear ;
end;
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    //如果选择【SELECT 语句】单选钮，让 DBGrid1 显示 Query1 查询形成的数据集
    DataSource1.DataSet :=Query1;  DBGrid1.DataSource :=DataSource1;
end;
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    //如果选择【其他数据操纵语句】单选钮，就让 DBGrid1 显示与 Table1 关联在一起的数据
    //通讯录表中的数据
    DataSource1.DataSet :=Table1;
    DBGrid1.DataSource :=DataSource1;
    Table1.Open ;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Query1.SQL.Clear ;           //清除 Query1 组件中原来的 SQL 语句
    Query1.SQL.add(Memo1.Text ); //添加当前输入的 SQL 语句
    Query1.Prepare ;           //优化
    if RadioButton1.Checked then //如果选择了【Select 语句】单选钮
        Query1.Open             //执行 SQL 语句，形成数据集并显示出来
    else                          //如果选择了【其他数据操纵语句】单选钮
        begin
            Query1.ExecSQL ;      //执行该 SQL 数据操纵语句
            Table1.Refresh ;      //刷新 Table1 组件，把执行的结果显示出来
        end;
    End;
end;

```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    Memo1.Clear;           //清除输入的语句
end;
procedure TForm1.Button3Click(Sender: TObject);
begin
    Query1.Close; Table1.Close;
    Application.Terminate; //结束应用程序的运行
end;
```

14.3 上机练习

14.3.1 上机练习一

【练习题目】：登录界面的设计

一般的软件系统都有登录系统用来验证访问者的身份（用户名和口令），这是一个比较通用的模块。请设计一个登录程序，完成的功能如下：合法的用户提供合法的口令就可进入系统，非法用户只能尝试三次，若不能够进入系统则应用程序自动结束。程序的设计界面如图 14-25 所示，程序的运行界面如图 14-26 所示。



图 14-25 程序设计界面

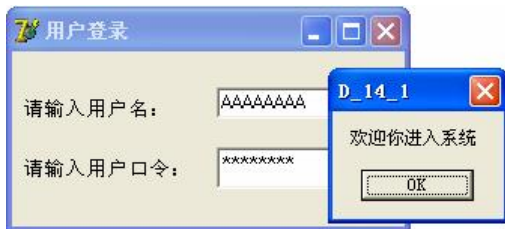


图 14-26 用户名和口令均输入正确时的程序运行界面

【要点提示】

(1) 用户信息的存放。可建立一个用户表，用来存入用户信息，用户表中应该包含用户名、口令、是否为管理员等信息。本题可建立一个 User 的表，它的结构如图 14-27 所示，其中字段 UserName 用来存放用户名，字段 Password 用户存放用户口令，字段 Administrator 用来表示该用户是否为管理员，值为 True 时表示是管理员，值为 False 时表示不是管理员。在表中添加一些用户信息作为模拟数据，模拟数据如图 14-28 所示。

(2) 功能实现。当用户输入完用户名后，程序自动在 User 表中的 UserName 字段中查找是否有该用户，如果没有该用户，则给出非法用户的提示信息，如果三次输入的用户名均不正确则应用程序自动终止。当用户名输入正确后，允许输入口令，口令输入后，程序自动把用户输入的口令与当前记录的 Password 字段值比较，若不相等则给出口令错误的提示信息，也一样允许用户输入三次，若三次均错误，则终止程序的执行。（注意：为便于讲解，本题无

论是输入用户名还是输入口令，输入结束后均需按 Enter 键确认。)

Field roster:				
	Field Name	Type	Size	Key
1	UserName	A	8	*
2	PassWord	A	8	
3	Admin	L		

图 14-27 User 表的结构

User	UserName	PassWord	Admin
1	AAAAAA	11111111	True
2	BBBBBB	22222222	False
3	CCCCCC	33333333	False
4	DDDDDD	44444444	True

图 14-28 User 表中的模拟数据

【参考代码】

```

var
  YHDL: TYHDL;
  YHM:String;           //全局变量 YHM 用来存放用户输入的用户名信息
  GLY:Boolean;          //全局变量 GLY 用来存放当前登录的用户是否为管理员信息,
                          //值为 True 时表示为管理员
  KL:String;            //全局变量 KL 用来存放用户输入的口令信息
implementation
var
  time:integer;          //用来存放用户尝试的次数
{$R *.dfm}
procedure TYHDL.FormCreate(Sender: TObject);
var
  curdir:string;         //curdir 存放应用程序所在目录
begin
  getdir(0,Curdir);      //获取当前目录
  Table1.DatabaseName :=Curdir;    //设置数据库名
  Table1.TableName:='User.db';      //设置表名为用户表
  Table1.Open ;             //打开表
  time:=0;                  //尝试次数为 0
end;
procedure TYHDL.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if ord(key)=13 then      //如果按 Enter 键
  begin
    Table1.IndexFieldNames :='UserName';    //设置索引字段名为'UserName'
    Table1.SetKey ;          //把用户表设置为查询状态
    Table1.FieldByName('UserName').AsString :=Edit1.Text ;
    //根据用户输入的用户名进行查找
    if Table1.GotoKey  then   //如果找到
    begin
      Edit2.SetFocus ;       //焦点设置在 Edit2，以便输入口令
      time:=0;              //重新设置 Time 值为 0，允许输入口令时也尝试三次
    end
  else
    //用户名输入错误
  
```

```

begin
    time:=time+1;           //尝试次数加 1
    if time<=2 then         //如果尝试次数小于 3
        ShowMessage('用户名非法, 请重输!')           //弹出提示
    else
        begin
            ShowMessage('你是非法用户, 不能进入系统! ');           //尝试次数超过 3 次
            Table1.Close;                                           //关闭表
            Application.Terminate ;                               //终止应用程序的执行
        end;
    end;
end;

end;
procedure TYHDL.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
    if Ord(Key) = 13 then
        if Table1.FieldByName('PASSWORD').AsString <> Edit2.Text then
            begin
                time := time + 1;
                if time < 3 then
                    begin
                        ShowMessage('口令错误, 请重新输入! ');
                        Edit2.Text := '';    Edit2.SetFocus;
                    end
                else
                    begin
                        ShowMessage('对不起, 你无法进入系统! ');    Close;
                    end
                end
            end
        else
            begin
                ShowMessage('欢迎你进入系统');
                YHM:=Table1.FieldByName('Username').AsString; //保存用户名到变量 YHM 中
                GLY:=Table1.FieldByName('Admin').AsBoolean;   //保存是否为管理员信息到 GLY 变量中
                KL:=Table1.FieldByName('Pass Word').AsString; //保存口令信息到变量 KL 中
                Table1.Close;                                   //关闭用户表
                YHDL.Hide ;                                    //隐藏登录界面
                // .....此处显示应用程序的主界面
            end;
        end;
end;
end;

```

14.3.2 上机练习二

【练习题目】：用 SQL 语句实现通讯录的浏览和维护

使用 SQL 语句实现通讯录的浏览和维护, 实现的功能与典型实例一基本相同, 但有以下

几点不同。

(1) 显示各字段的内容的组件不再使用【Data Controls】选项卡上的组件,而是使用像 TEdit、TComboBox、TMemo 等的一般组件。

(2) 各按钮实现的功能基本上与上一个实验相同,不同之处只是【增加】、【删除】、【修改】等操作均要使用 SQL 语句来实现。

(3) 为了能够执行 SQL 语句,本题增加了一个 TQuery 组件用来执行 SQL 语句。

(4) 为了能够使表中数据的变化能够立即显示出来,本题增加了一个 TDBGrid 组件,用来显示通讯录表的内容。

本题的程序设计界面如图 14-29 所示,程序的运行界面如图 14-30 所示。

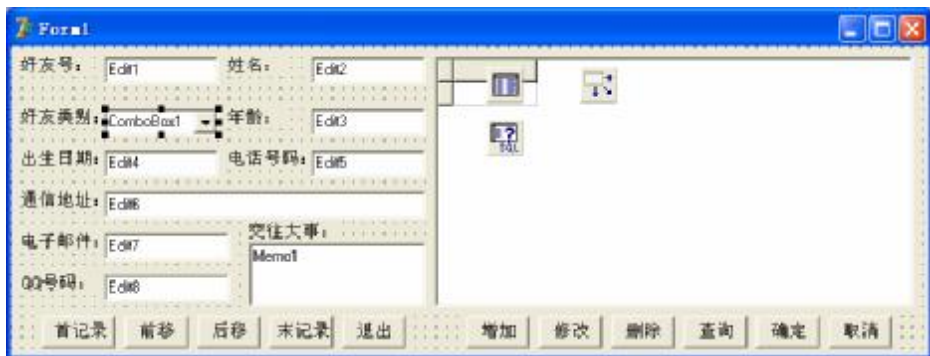


图 14-29 程序设计界面



图 14-30 程序运行界面

【要点提示】

(1) 由于不是采用数据控制组件显示数据库中的内容,因此每当记录指针的位置发生了变化,应该要把当前记录中的数据显示在相应组件中。

(2) 本题使用 TDBGrid 显示通讯录表中的数据,当用户在该表上单击时可能会改变当前记录的位置,因此应当使用当前记录的值得来更新各显示组件。

(3) 为执行 SQL 语句,本题使用了 TQuery 组件,使用该组件执行 SQL 语句的一般方法是:首先执行 TQuery 组件的 SQL.Add 方法把要执行的 SQL 语句作为字符串加进来;如果 SQL 语句中带有参数,应该通过 TQuery 组件的 Params[] 给 ParamByName[] 数组属性的参数赋

值；通过调用 TQuery 组件的 ExecuteSQL 方法或 Open 方法来执行相应的 SQL 语句。

(4) “插入”操作可使用 INSERT 语句、“修改”操作可使用 UPDATE 语句，“删除”操作可使用 DELETE 语句。删除操作需要条件，否则删除的是全部记录，为实现当前记录的删除，可先用一个变量记下当前记录的“好友号”字段的值，然后在“删除”语句中设置条件为“好友号等于该变量的值”。

【参考代码】

```
implementation
var
    BM:TBookMark;           //该变量用于在添加记录时把当前记录位置作为一个书签
    AddOrNot:Boolean;       //该变量用来表示是添加操作还是修改操作，True 表示添加
{$R *.dfm}
procedure DispValue();      //在编辑框、组合框或 Memo 组件中显示当前记录的内容
begin
    Form1.Edit1.Text :=Form1.Table1.fieldbyname('好友号').asString;
    Form1.Edit2.Text :=Form1.Table1.fieldbyname('姓名').asString;
    Form1.ComboBox1.Text :=Form1.Table1.fieldbyname('好友类别').asString;
    Form1.Edit3.Text :=Form1.Table1.fieldbyname('年龄').asString;
    Form1.Edit4.Text :=Form1.Table1.fieldbyname('出生日期').asString;
    Form1.Edit5.Text :=Form1.Table1.fieldbyname('电话号码').asString;
    Form1.Edit6.Text :=Form1.Table1.fieldbyname('通讯地址').asString;
    Form1.Edit7.Text :=Form1.Table1.fieldbyname('电子邮件').asString;
    Form1.Edit8.Text :=Form1.Table1.fieldbyname('QQ 号码').asString;
    Form1.Memo1.Text :=Form1.Table1.fieldbyname('交往大事').asString;
end;
procedure ClearValue();     //消除在编辑框、组合框或 Memo 组件中显示的内容
begin
    Form1.Edit1.Text :='';   Form1.Edit2.Text :='';
    Form1.ComboBox1.Text:= '';   Form1.Edit3.Text :='';
    Form1.Edit4.Text :='';   Form1.Edit5.Text :='';
    Form1.Edit6.Text :='';   Form1.Edit7.Text :='';
    Form1.Edit8.Text :='';   Form1.Memo1.Text :='';
end;
//省略了 ButtonAddEdit 和 ButtonOkCancel 两个过程，参见典型实例一
procedure Cannotedit();     //用来设置各输入的数据控制组件不可输入
begin
    Form1.Edit1.ReadOnly:=True;   Form1.Edit2.ReadOnly:=True;
    Form1.ComboBox1.Enabled :=False;   Form1.Edit3.ReadOnly:=True;
    Form1.Edit4.ReadOnly:=True;   Form1.Edit5.ReadOnly:=True;
    Form1.Edit6.ReadOnly:=True;   Form1.Edit7.ReadOnly:=True;
    Form1.Edit8.ReadOnly:=True;   Form1.Memo1.ReadOnly:=True;
end;
procedure Canedit();        //用来设置各输入的数据控制组件可以输入
begin
```

```

        Form1.Edit1.ReadOnly:=False;    Form1.Edit2.ReadOnly:=False;
        Form1.ComboBox1.Enabled:=True;    Form1.Edit3.ReadOnly:=False;
        Form1.Edit4.ReadOnly:=False;    Form1.Edit5.ReadOnly:=False;
        Form1.Edit6.ReadOnly:=False;    Form1.Edit7.ReadOnly:=False;
        Form1.Edit8.ReadOnly:=False;    Form1.Memo1.ReadOnly:=False;
    end;
    .....//此处省略了 DisabledButton 过程, 参见典型实例一
procedure TForm1.FormCreate(Sender: TObject);
var
    Curdir:string;
begin
    .....//省略的部分同典型实例一
    Disp Value;
    Query1.DatabaseName:=Curdir;          //设置 Table1 的 Databasename 为当前目录
end;
procedure TForm1.Button1Click(Sender: TObject); //首记录
begin
    .....//省略的部分同典型实例一
    Disp Value;
end;
procedure TForm1.Button2Click(Sender: TObject); //前移
begin
    .....//省略的部分同典型实例一
    Disp Value;
end;
procedure TForm1.Button3Click(Sender: TObject); //后移
begin
    .....//省略的部分同典型实例一
    Disp Value;
end;
procedure TForm1.Button4Click(Sender: TObject); //末记录
begin
    .....//省略的部分同典型实例一
    Disp Value;
end;
    .....//省略了 Button5Click 事件过程, 参见典型实例一
procedure TForm1.Button6Click(Sender: TObject); //添加
begin
    ClearValue; BM:=table1.GetBookmark ;          //记下当前记录位置
    CanEdit;                                         //允许输入
    ButtonAddEdit;                                   //设置按钮状态
    AddOrNot:=True;                                  //是增加操作
end;
    procedure TForm1.Button7Click(Sender: TObject); //修改
begin

```

```

AddOrNot:=False;                                //不是增加操作
CanEdit; ButtonAddEdit;
end;
procedure TForm1.Button8Click(Sender: TObject);    //删除
var
    hyh:string;
begin
    if Application.MessageBox('真的要删除吗?','删除提示框',MB_OKCANCEL)=IDOK then
    begin
        hyh:=Table1.FieldByName('好友号').AsString;
        Query1.SQL.Clear;
        Query1.SQL.Add('Delete From TXL WHERE 好友号=(:A1)');
        Query1.Params[0].AsString:=hyh;           QUERY1.ExecSQL;
        Table1.Refresh;        Disp Value;
    end;
end;
.....//省略了 Button9Click 事件过程, 参见典型实例一
procedure TForm1.Button10Click(Sender: TObject);   //确定操作
var
    hyh:string;
begin
    If (length(trim(Edit1.text))=0) then           //好友号为空
    begin
        ShowMessage('必须输入好友号');          edit1.SetFocus;
    end
    else
    begin
        if AddOrNot then
        begin
            Query1.SQL.Clear;
            Query1.SQL.Add('INSERT INTO 通讯录(好友号,姓名,好友类别,年龄,出生日期,
                电话号码,通讯地址,电子邮件,QQ 号码,交往大事)');
            Query1.SQL.Add('Values (:(P1),:(P2),:(P3),:(P4),:(P5),:(P6),:(P7),:(P8),:(P9),:(P10))');
        end
        else
        begin
            hyh:=Table1.fieldbyname('好友号').AsString;
            Query1.SQL.Clear;
            Query1.SQL.Add('Update 通讯录 SET 好友号=(:P1),姓名=(:P2),好友类别
                =(:P3)');
            Query1.SQL.Add(',年龄=(:P4),出生日期=(:P5),电话号码=(:P6)');
            Query1.SQL.Add(',通讯地址=(:P7),电子邮件=(:P8),QQ 号码=(:P9),交往大
                事=(:P10)');
            Query1.SQL.Add('WHERE 好友号=(:P11)');
            Query1.Params[10].AsString:=hyh;
        end
    end
end;

```


4. 要使 TQuery 组件中的 SQL 语句执行后返回一个结果数据集, 应调用 TQuery 组件的_____方法。

A. Add

B. Open

C. ExecSQL

D. Open 和 ExecSQL

二、填空题 (40 分, 每空 5 分)

1. TDataSource 组件是通过_____属性与 TTable 组件建立联系的。

2. 数据控制组件要访问数据集中的某个字段, 首先应通过设置它的_____属性以便和 TDataSource 组件建立联系, 然后应设置它的_____属性以便和具体的字段建立联系。

3. 当记录指针位于_____位置时, TTable 组件的 Eof 属性为 True, 当记录指针位于_____位置时, TTable 组件的 Bof 属性为 True。

4. 调用 TQuery 语句的 SQL 属性的_____方法可以清除掉 SQL 属性中存放的字符串。

5. 调用 TQuery 语句的 SQL 属性的_____方法可以把 SQL 语句增加进它的 SQL 属性中。

6. 要实现参数查询, 可以在 TQuery1 组件中增加包含参数的 SQL 语句, 为了给参数赋值, 应使用 TQuery 组件的_____数组属性。

三、程序设计题 (40 分)

首先建立一个 TSXX 的表, 表的结构如图 14-31 所示。注意在定义表结构时, 建立了两个索引, 索引名为 BOOKNAME (以书名作为索引字段) 和 BOOKNO (以书号作为索引字段)。然后在表中输入十条以上的数据, 如图 14-32 所示。表结构建立成功并输入数据后, 设计一个对该表进行浏览和管理的程序, 程序的设计界面如图 14-33 所示, 程序的运行界面如图 14-34 所示。要求使用以下方法来实现程序的功能: (1) 显示各字段内容的组件不再使用 Data Controls 选项卡上的组件, 而是使用像 TEdit、TCheckBox 等的一般组件; (2) 查询使用 FindKey 方法来实现。

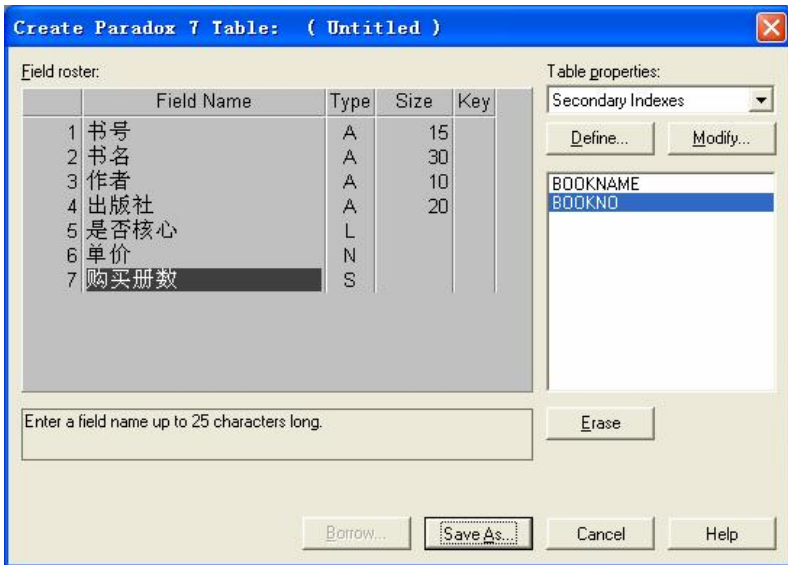
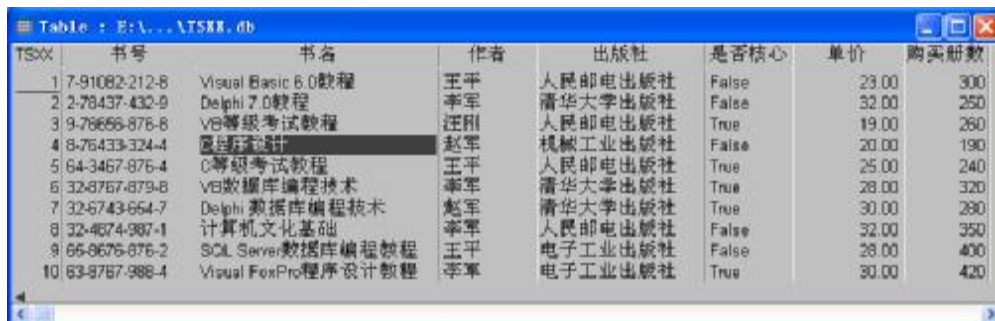


图 14-31 TSXX 表的结构



TSXX	书号	书名	作者	出版社	是否核心	单价	购买册数
1	7-91082-212-8	Visual Basic 6.0教程	王平	人民邮电出版社	False	23.00	300
2	2-78437-432-9	Delphi 7.0教程	李军	清华大学出版社	False	32.00	250
3	9-78656-876-8	VB等级考试教程	汪刚	人民邮电出版社	True	19.00	260
4	8-76433-324-4	C语言设计	赵军	机械工业出版社	False	20.00	190
5	64-3467-876-4	C等级考试教程	王平	人民邮电出版社	True	25.00	240
6	32-8767-879-8	VB数据库编程技术	李军	清华大学出版社	True	28.00	320
7	32-6743-664-7	Delphi 数据库编程技术	赵军	清华大学出版社	True	30.00	290
8	32-4674-987-1	计算机文化基础	李军	人民邮电出版社	False	32.00	350
9	66-8676-876-2	SQL Server数据库编程教程	王平	电子工业出版社	False	28.00	400
10	63-9787-988-4	Visual FoxPro程序设计教程	李军	电子工业出版社	True	30.00	420

图 14-32 TSXX 表的内容



图 14-33 程序设计界面



图 14-34 查询时的程序运行界面